Examensarbete
LITH-ITN-ED-EX--06/018--SE

Development software for stabilization platform

Jonas Nilsson

2006-05-17

Linköpings universitet
TEKNISKA HÖGSKOLAN

Department of Science and Technology
Linköpings Universitet
SE-601 74 Norrköping, Sweden

Institutionen för teknik och naturvetenskap
Linköpings Universitet
601 74 Norrköping

LITH-ITN-ED-EX--06/018--SE

Development software for stabilization platform

Examensarbete utfört i elektronikdesign
vid Linköpings Tekniska Högskola, Campus Norrköping

Jonas Nilsson

Handledare Christian Walsh
Examinator Ole Pedersen

Norrköping 2006-05-17

Avdelning, Institution
Division, Department

Institutionen för teknik och naturvetenskap

Department of Science and Technology

Datum
Date

2006-05-17

Språk
Language

☐ Svenska/Swedish
☒ Engelska/English

☐ _____

Rapporttyp
Report category

Examensarbete
☐ B-uppsats
☐ C-uppsats
☒ D-uppsats

☐ _____

ISBN

ISRN **LITH-ITN-ED-EX--06/018--SE**

Serietitel och serienummer
Title of series, numbering

ISSN

URL för elektronisk version

Titel
Title

Development software for stabilization platform

Författare
Author

Jonas Nilsson

Sammanfattning
Abstract

I detta examensarbete beskrivs modifikationen av redan befintlig mjukvara för att passa den stabiliserande plattformen. Den stabiliserande plattformen har som syfte att simulera ett fordon i rörelse. Med hjälp av fordons simulatorn uppnår företaget möjligheter att idag utföra interna realistiska tester vid utveckling av ny hård och mjukvara för stabiliserande system. Plattformen erhåller sina indata från vägprofilgeneratorn som är ett PC baserat simulationsprogram programmerad i LabVIEW. I vägprofilgeneratorn finns möjligheten att skapa egna vägprofilsignaler eller använda redan tidigare inspelade signaler från verkliga tester. Examensarbetet utfördes i samarbete med Curtiss-Wright Antriebstechnik GmbH i Neuhausen am Rheinfall, Schweiz

Nyckelord
Keyword

DRUi 360, RS-485, Communication, DSP, Closed loop, PI, Regulator, Filter, USB, Ground Profile Generator, Software, LabVIEW, Code Composer Studio 2.0, Simulation software, TMS320LC54x

LINKÖPING UNIVERSITY
ELECTRONIC PRESS

Upphovsrätt

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

Copyright

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© Jonas Nilsson

Development Software for Stabilization Platform

In cooperation with Curtiss-Wright Antriebstechnik GmbH
Neuhausen am Rheinfall, Switzerland

Jonas Nilsson

May 2006

Abstract

This Master thesis describes the development and modification of already existing assembler software for the DRUi 360. The DRUi is a controller unit for stabilization tasks mostly used in the military industry. The DRUi controls the stabilization platform which has the purpose to simulate a moving vehicle. The aim with this platform is for internal development of new hardware and software, evaluate new ideas and components for stabilization systems. The platform is also an internal education system for the engineers. The DRUi receives the input reference signal from the Ground Profile Generator which is a graphical interface running on a PC were the user has the ability to create or send already recorded road signals from a moving vehicle to the driver unit. In this case the DRUi.

The main used software's for this project are Code Composer Studio for programming the DRUi software and LabVIEW which is a graphical programming language used to program the Ground Profile Generator.

A theoretical study of motion control is performed together with getting familiar of al the equipment.

The new software for the DRUi is adapted and implemented. Filter design for the regulator unit and necessary measurements on the test stand are presented together with the results of the production run of the stabilization platform.

The stabilization platform is not fully running when this thesis ended. When the production test was performed disturbances appeared in the system. The reasons for these disturbances are not yet discovered. Al safety routines needed for running the platform secure and prevent system damage are implemented to the system software.

Preface

This report is a result of a six months long thesis work, made in cooperation with the company Curtiss-Wright Antriebstechnik GmbH located in Neuhausen am Rheinfall Switzerland. The thesis describes the programming and adaptation of already existing software for a stabilization platform with aiming tasks. The platforms main function is to work as an internal test system for software and hardware development as well as internal education.

Jonas Nilsson

Neuhausen am Rheinfall, March 2006

Thank you!

I would like to thank Mr Harry Zai for answering my email and letting me come for an interview and giving me this opportunity and great experience during my six months at Curtiss-Wright. I am very grateful for that. Many thanks to my instructor Christian Walsh your skills are a great. Thanks for the support you all gave me when you had so much to do Klaus Hofacker, Reto Gätcher, and Roman Rainer. And thanks allot to the rest of the staff at engineering department and every one which I been in contact with. You all have been very friendly!

Contents

1 INTRODUCTION ***9***

1.1 Purpose . . . 9
1.2 Background . . . 10
1.3 Method . . . 10
1.4 Company presentation . . . 11
1.5 Limitations . . . 11
1.6 Thesis structure . . . 12

2 THEORY ***13***

2.1 Motion Control . . . 13
2.2 Programmable Motion Control . . . 13
2.3 Structure . . . 14
2.4 PI Regulator . . . 15
2.5 System overview . . . 16
2.6 Ground profile generator . . . 17
2.7 Communication . . . 19
2.7.1 RS-485 . . . 20
2.8 DRUi regulator structure . . . 20

3 SYSTEM SETUP ***22***

3.1 Test stand . . . 22
3.2 DRUi 360 . . . 23
3.3 CPU . . . 24
3.3.1 Architecture . . . 24
3.3.2 Features of c54x DSP . . . 26
3.3.3 Addressing modes . . . 26
3.3.4 Serial ports . . . 27
3.4 Zilog comunication unit . . . 27
3.4.1 Features of the USC . . . 28
3.5 USB-COMi-M . . . 28
3.5.1 Features of USB-COMi-M . . . 29

3.6 XDS510 USB Emulator . . . 29
3.7 AC/DC servo motor . . . 29
3.7.1 Motor feedback . . . 30
3.8 Softwares . . . 31
3.8.1 LabVIEW . . . 31
3.8.2 Code Composer Studio 2.0 . . . 32
3.8.3 Programming the DSP . . . 33

4 DEVELOPMENT SOFTWARE ***35***
4.1 Setting up the software . . . 35
4.2 Understanding the software . . . 35
4.2.1 Flow diagram . . . 36
4.3 Code structure . . . 36
4.3.1 Communication . . . 37
4.3.2 Ground profile generator communication . . . 38
4.3.3 Input and output buffer . . . 39
4.3.4 Timing received and processed values . . . 39
4.3.5 Counting the input values . . . 39
4.3.6 Processing the values . . . 40
4.3.7 Speed and position mode . . . 40
4.4 Safety routines . . . 40
4.4.1 Nlimiter . . . 41
4.4.2 Primilim and lower speed . . . 42
4.4.3 Aimbit and stdbybit . . . 43
4.4.4 Stop command . . . 43
4.4.5 Brake . . . 44
4.4.6 Zurrcon . . . 44
4.5 Debugging the software . . . 44
4.5.1 Timing debug . . . 44
4.5.2 Debug buffer . . . 44
4.5.3 XF-flag . . . 45
4.5.4 Braking the current loop . . . 45
4.5.5 Flag for last processed value . . . 46
4.5.6 DRUi Sending Data . . . 47
4.5.7 Disabled BITF . . . 48
4.6 Test stand measurements . . . 49
4.6.1 Teaching the system . . . 50
4.6.2 Measuring the unbalance . . . 52
4.6.3 Setting up the filters . . . 53
4.6.4 Filters . . . 54
4.7 Running the test stand . . . 56

4.7.1 New boot software 56
4.7.2 Disturbance 56
4.8 Debugging at the test stand 58

5 Results ***59***
5.1 Conclusions and discussion 59
5.2 Further work 60

A Code flow diagram ***63***

B Assembler code ***77***

List of Figures

1.1 System structure 9
1.2 Drawing test-stand 12

2.1 Closed loop system 15
2.2 PI regulator structure 16
2.3 Development system 17
2.4 Ground profile generator 18
2.5 LabVIEW simulation window 19
2.6 Regulator structure DRUi 20

3.1 Stabilization platform 23
3.2 DRUi 360 24
3.3 Architecture Texas Instruments c54x DSP 25
3.4 Architecture Zilog Serial Controller 28
3.5 USB-COMi-M 29
3.6 Motor parts 30
3.7 LabVIEW block diagram 32
3.8 Code Composer Studio 33

4.1 Syscon structure 37
4.2 Input values 38
4.3 Nsoll Limiter 41
4.4 Principle of NLIMITER 41
4.5 Primitiv Limiter And Limiter 50% 42
4.6 PRIMILIM and LOWER_SPEED principle 43
4.7 Buffer actual and demanded values 45
4.8 Current loop 46
4.9 Output values with disturbance 46
4.10 PC-DRUi Communication 47
4.11 Current disturbance 48
4.12 BITF active 48

4.13 BITF disabled 49
4.14 DACSView measurement menu 50
4.15 Primilim setup 51
4.16 Unbalance 52
4.17 FFT window 53
4.18 FFT Measurement 54
4.19 Notch filter 1 55
4.20 Notch filter 2 55
4.21 Square wave input 57
4.22 Sinus input 57
4.23 Sinus with disturbance 58

Abbreviations

BPS Bytes Per Second

BLDC Bruchless DC Motor

CIF Customer User Interface

CPU Central Processing Unit

DC Direct Current

DMA Direct Memory Access

DSP Digital Signal Processor

DRUi Drive Unit Intelligent

FFT Fast Fourier Transform

FIFO First In First Out

IST Actual

PI Proportional Integral

PMC Programmable Motion Control

PMSM Permanent Magnet Synchronous Motor

SOLL Demanded

RPM Revolutions Per Minute

TDM Time-Division Multiplexed

USC Universal Serial Controller

Chapter 1

INTRODUCTION

1.1 Purpose

This thesis primary object is to adapt and configure already existing software for the DRUi 360. Normally the DRUi communicates with an AXCON. The AXCON is a hardware controlled communication unit containing a CIF-card ”Customer User Interface”. In this project the DRUi communicates directly with the PC over a serial port with full-duplex RS485 communication. The DRUi is the controller unit that will be used to regulate the test platform. By sending a signal to the DRUi it regulates the platform to move similar as the given input signal. The signal could be either an recorded road characteristic signal from a moving vehicle, a sinus wave, pure offset or what the user needs.

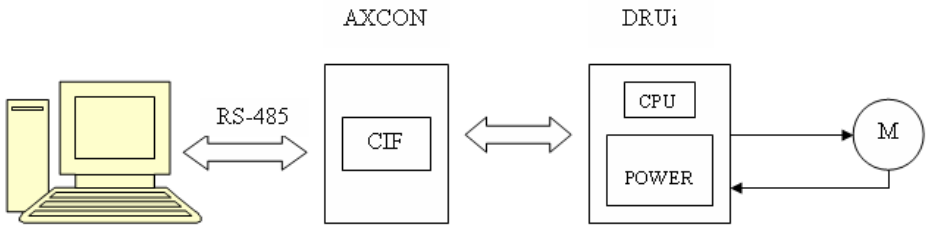


Figure 1.1: System structure

1.2 Background

The new platform has the purpose for the company to reach a more rapid and accurate development phase of new hardware and software. New ideas and components could easily be implemented and evaluated at the test rig. Similar components from different manufacturers could easily and accurate be compared and evaluated due to performance and costs. The platform gives a great opportunity for new engineers to practise and obtain knowledge about stabilization systems. In the end the platform hopefully decrease the research and evaluation costs of developing new stabilization systems.

1.3 Method

- First an orientation about the area motion control was done. Not so deep and theoretical, just only a brief orientation about the subject. How different regulators were used and how a closed loop system performs. What types of different system feedbacks normally are used in a regulator system. To get a better understanding of the complete system some small tests were done to see what happens and what type of effects different adjustments has on the system. That helped also to get familiar with the programming software's Code Composer Studio 2.0 and LabVIEW.

- Next step were to analyse the hardware and especially the software for the DRUi that processes the given commands from the PC. By structuration and analyses of the software, suitable reconfigurations of the Ground Profile Generator and the software for programming the DRUi are done.

- At least the whole system was mounted at the stabilization platform and measurements on the platform take part. The system had to be teached were the platforms end position are to prevent the system of running in to the metal end stops. Measuring of the system unbalance was performed together with a frequency analysis of the regulator unit "DRUi". By the frequency response of the measurements suitable filters are designed to get an acceptable performance by the system. Then at least a production run of the system were performed when all the functions were implemented.

1.4 Company presentation

January the first 1999 SIG Antriebstechnik AG were taken over by Curtiss-Wright Corporation and renamed to Curtiss-Wright Antriebstechnik GmbH (CWAT).

CWAT belongs to the Business Unit: Curtis Wright Controls Engineered Systems. Curtiss-Wright is divided into the three different areas, Motion Control, Flow Control and Metal Treatment.

The facilities in Neuhausen has approximately 80 employees were 50% are engineers (Electronics and Mechanics) and belongs to the area Motion Control. The main developments in Neuhausen are drive systems for industry, defence applications and vehicles in the area of electronics, software and system integration, electro-mechanics and electro-hydraulic products. The markets are mainly in Europe but also in North-America, Asia and South Africa[1].

Curtiss-Wright Corporation has a long history with its roots dating back to the Wright brother's first flight in 1903. The company is since 1929 listed at the NYSE New York Stock Exchange. The number of employees is today more than 5900 around the world.

1.5 Limitations

This project only cover the lover part of the test stand, with the purpose of simulate a moving vehicle. The upper part of the test stand has stabilization tasks with feedback from gyro. That part is not included in this thesis work. The lower circle shows the lower part of the system. The upper circle show the tower which will be stabilized due to its reference signal from a gyro.

[1] www.cwat.com

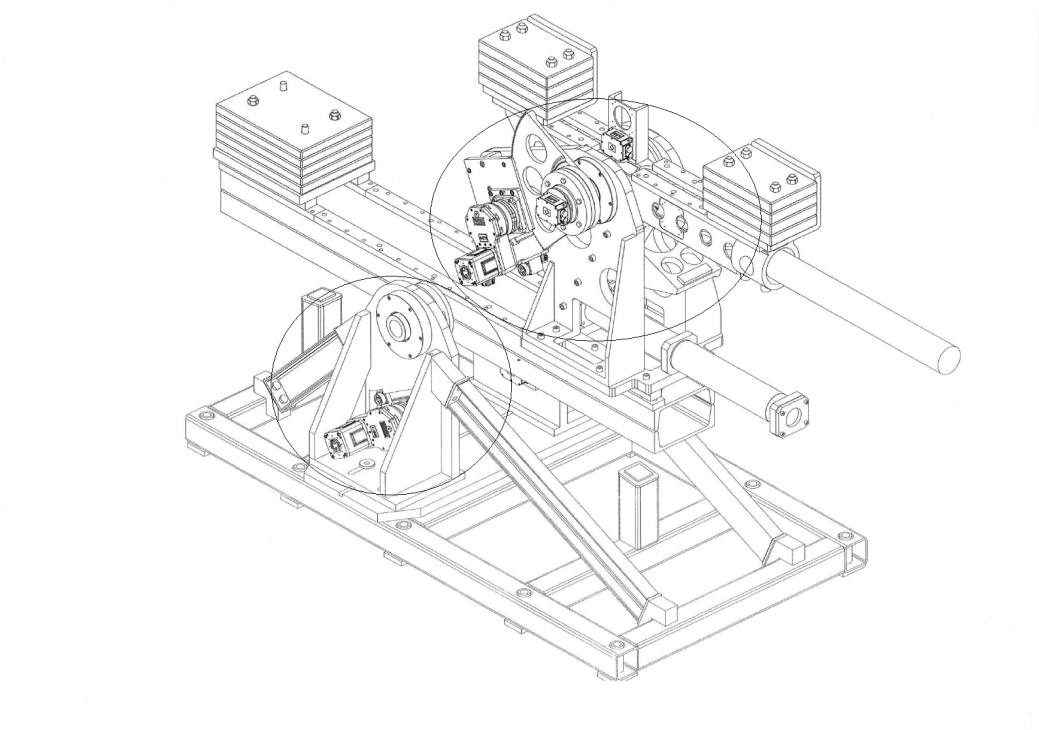


Figure 1.2: Drawing test-stand

1.6 Thesis structure

The first chapter, THEORY describes the concept for motion control, explaining the closed loops and regulator structures. In this chapter the used software and the assembler programming language are also in brief described.

The chapter SYSTEM SETUP describes the system hardware, the programming structure and the mechanical test stand itself with its features.

The third chapter DEVELOPMENT SOFTWARE describes the work made on the system and all the measurements.

The thesis ends with the chapter RESULT that describes the result and problems during the thesis work. It also discusses what could be done different and possible improvements for future work.

Chapter 2

THEORY

For better understanding of this thesis the next chapter will describe the basic concepts of motion control. It describes the theory behind the regulators and explains the structure of the complete system. It also describes the programming language and the main used software's for this project.

2.1 Motion Control

There are many different types of motion controls. In centuries back the control of position and speed where done mechanical by changing gears, using series of cams or hydraulic or pneumatic cylinders to control processes in the textile and other various industries. The early automated system where highly dedicated for one specific solution and demanded lots of effort for retooling, even though small changes in the process. Therefore early automation started in the textile industry were same operation are repeated over and over again[1].

2.2 Programmable Motion Control

Today almost every process were motion control are needed the system are realized through Programmable Motion Control (PMC) which is defined by a programmable hardware and software in a micro controlled system. Often the system is based on a DSP or a PIC-processor within a controller box. The system is than easily reconfigured through small changes in the program code and its peripheral components.

[1]Motion Control Handbook

2.3 Structure

A normal micro controlled system has some input sensor devices and a system feedback controlling either a linear or a rotary motion. The input signal comes from a host or some kind of interface, for example a connection from a PC.

The amplifier receives its signals from the controller and boosts the signals to a level adjusted for the actuator. In this project the actuator is a DC servo-motor, it is the DC servo-motor that gives the system its physical motion. The behaviour of the motor is close connected to the physical design of the amplifier, which is the speed/position driver of the actuator. A motion controlled system has in many ways the purpose to control either one or a combination of different parameters such as, Torque, Acceleration, Position and Speed[2].

A normal micro controlled system has some input sensor devices and a system feedback controlling either a linear or a rotary motion. The input signal comes from a host or some kind of interface, for example a connection from a PC.

The amplifier receives its signals from the controller and boosts the signals to a level adjusted for the actuator. In this project the actuator is a DC servo-motor, it is the DC servo-motor that gives the system its physical motion. The behavior of the motor is close connected to the physical design of the amplifier, which is the speed/position driver of the actuator. A motion controlled system has in many ways the purpose to control either one or a combination of different parameters such as, Torque, Acceleration, Position and Speed[3].

The system feedback depends on the actual measurements and which type of control elements that are used. Common feedback signals are the same as described above and other signals as Temperature, Current and Voltage

This type of structure with feedback is a closed loop system.

[2]Motion Control Handbook
[3]Motion Control Handbook

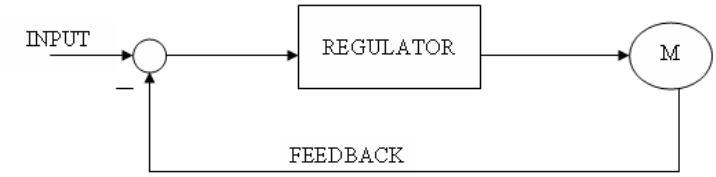


Figure 2.1: Closed loop system

2.4 PI Regulator

The closed loop system is controlled by PI-regulators. To decouple the torque, current or speed, it is necessary to engage several mathematical transforms, and this is where the DSP adds the most value. The processing capability provided by the DSP can enable these mathematical transformations to be carried out very quick almost in real time[4]. This in turn means that the entire algorithm controlling the motor can be executed at a very high rate. PI stands for Proportional and Integrational. This is the two distinctive elements of a PI-controller. The proportional term of the controller reacts on changes in the reference signal and therefore reacts more rapidly than the Integrational part[5]. If the proportional part in the regulator is given to large value the system could easily start to self oscillate.

$$K_P = Proportionalterm = K(e(t))$$
$$K_i = Integrationalterm = K(\frac{1}{T_i}\int_0^\tau e(\tau)d\tau)$$
$$u(t) = K(e(t) + \frac{1}{T_i}\int_0^\tau e(\tau)d\tau)$$

Where $e(t)$is the regulator error" input - feedback" which gives the p-part and the integral part $\frac{1}{T_i}\int e(\tau)d\tau)$. The integral part is changing as long as the actual value differs from the demanded value.

[4]Motion Control Handbook
[5]Proportional & Integral Controllers

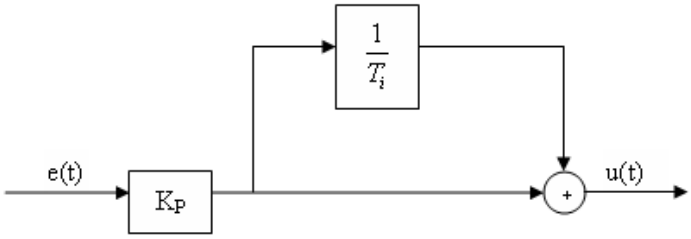


Figure 2.2: PI regulator structure

2.5 System overview

The control system can be divided into three important parts.

- The PC containing the ground profile generator.
- The communication between PC and the DRUi.
- The DRUi itself connected to the DC servo-motor within the closed loop system.

The PC sends speed or position commands to the DRUi. The DRUi regulates the DC servo-motor by sending demanded position, speed and current. The DC servo-motor present its feedback from the encoder mounted on the back of the DC-servo motor. The encoder gives information of position, current and the actual speed of the servo motor. The actual value from the DC servo-motor is then presented in the graphical window together with the demanded value in simulation menu of the Ground Profile Generator. The system control loop of the speed and position regulators has an update rate of 2 kHz. The update rate of the control loop is much faster than the processing of input data values. It is necessary for the accuracy and speed, which means time to compensate within the PI regulator. This is the control and communication part of the system that regulates the test stand.

Figure 2.3: Development system

Figure 2.3 shows the development system. with PC, DRUi and DC servo-motor.

2.6 Ground profile generator

The ground profile generator is a graphical user interface programmed in the development tool LabVIEW from National Instruments. The ground profile generator is a interface for create and send simulated signals or recorded road signals to the DRUi. The ground profile generator has three essential menus.

- Road Characteristics
- Simulation
- Communication

Figure 2.4: Ground profile generator

The road characteristics ”Fahrweg-Egenschaften” are the menu for loading an already recorded signal ”from a moving vehicle” or create your own signal, for example a sinus or a saw tooth signal.

The simulation menu is where the actual simulation appears, the signal is loaded and the system is ready to proceed the simulation. There are also functions to let the user change period time of send cycle, number of values to be sent. The sampled input signal is presented graphically together with the signal from the DRUi presenting the actual speed feedback from the DC servo-motor. Figure 2.5 shows the simulation window with its reference signal and the actual position of the motor

The Communication menu is just for testing and synchronizes the communication between PC and DRUi and let the user to change communication port. The DRUi sends back an answer code if the synchronization is okay, otherwise an error code appears in the graphical window. Figure 7 shows the simulation window with its demanded and actual speed. The output signal

is shifted by 100 values and therefore making the delay between the signals presented in the graphical window.

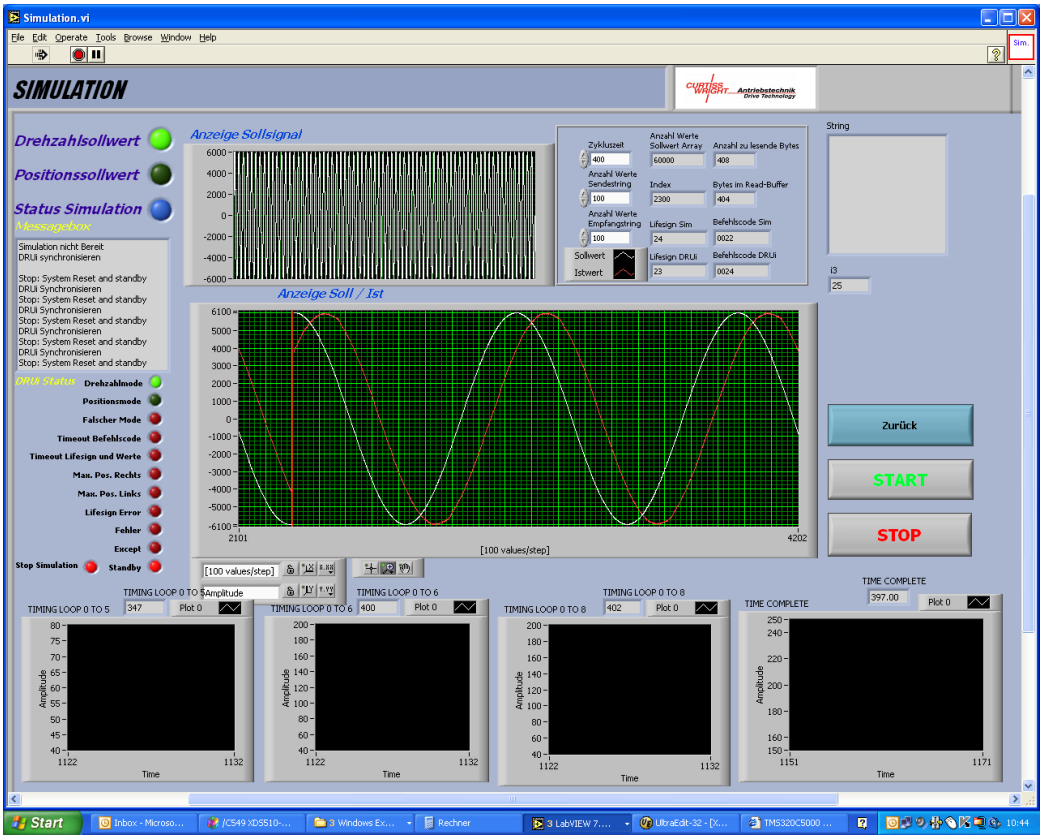


Figure 2.5: LabVIEW simulation window

2.7 Communication

The ground profile generator samples the signal and store the values in packages containing 100 values each.

Every value contains: 20bit (16 + 2 start + 2 stop)bit. Every data package contains 20bit Command (16 + 2start + 2stop)bit and 20bit Lifesign (16 + 2start + 2stop)bit. The lifesign is a counter for every new data package.

That gives that every data package contains $20 + 20 + (100 * 20) = 2040$ bit per send cycle. With a send cycle of 500ms the PC sends two packages every second, which result in a transfer rate of 4080bit/s. The generator is not sending a new package until it receives an answer from the DRUi. So if the communications fail, the DRUi halts. In the initialization phase The DRUi answer directly with 100 zeros to get a new data package of 100 values.

2.7.1 RS-485

The communication between PC and DRUi are 4-wire point-to-point serial communication with full-duplex RS485 at 115.2k bps. With full-duplex means communication of data in both directions simultaneously. RS-485 is not a standard in ordinary home PC:s though it using the COM-port that every standard PC has. Normally the comport uses the standard 2-wire RS-232 interface. The RS-485 is more suitable for data acquisition then RS-232 and do not have that many limitations as the standard RS-232 serial communication have. The RS-485 supports more drivers and receivers, up to 32 each[6]. The RS-485 communication is widely used in industrial applications.

2.8 DRUi regulator structure

The regulator structure of the DRUi consists of three major closed loops which is the Current loop, Speed loop and Position loop. The PI-regulators of the Speed and Position loop runs with a clock frequency of $2kHz$. The regulator of the Current loop runs with the system clock of $20kHz$. To run the system in speed mode the position loop are disconnected and the input offset values are connected to NSOLL. SOLL ”Demanded”, IST ”Actual”. The information about speed and position are transfered from the encoder mounted on the back of the AC/DC motor. Se figure 3.6 for encoder emplacement.

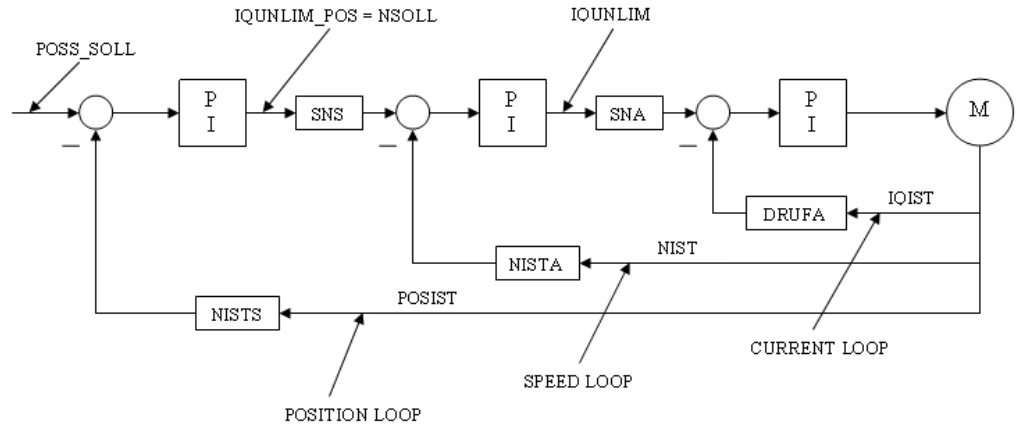


Figure 2.6: Regulator structure DRUi

[6]RS 422 and 485 Standards.

SNS and SNA is scaling factors. NISTA, NISTS and DRUFA are the system feedback factors.

Chapter 3

SYSTEM SETUP

In this chapter the system hardware will be described and explained. First the test stand will be introduced to get the reader familiar with the platform and how the detailed componets are placed in the system The heart of the system is the DRUi. It is there the processing and regulation of speed commands are done through the CPU. The CPU is a DSP (Digital Signal Processor) from Texas instruments. The Communication part that receives it information from the PC are handled by the Universal Serial Controller Z16C30 from Zilog. The containing parts of this chapter is the complete system which regulates and gives the system its physical motion.

3.1 Test stand

The test stand can be divided into two parts, upper and lower. The lower part is the vehicle part with the purpose to simulate a moving vehicle. The upper part is the armoury part for the weapon stabilization task. By sending the road signals from the Ground Profile Generator the DRUi process and controls the stand. It compensates the actual position compared to the demanded position from the input signal. The DRUi then regulates the current so the stand starts to move and following the input signal as close as the regulator structure manage to regulate the system. The stand has only the possibility to move around one centre axis. It could not lurch sideways as a vehicle would move in the reality.

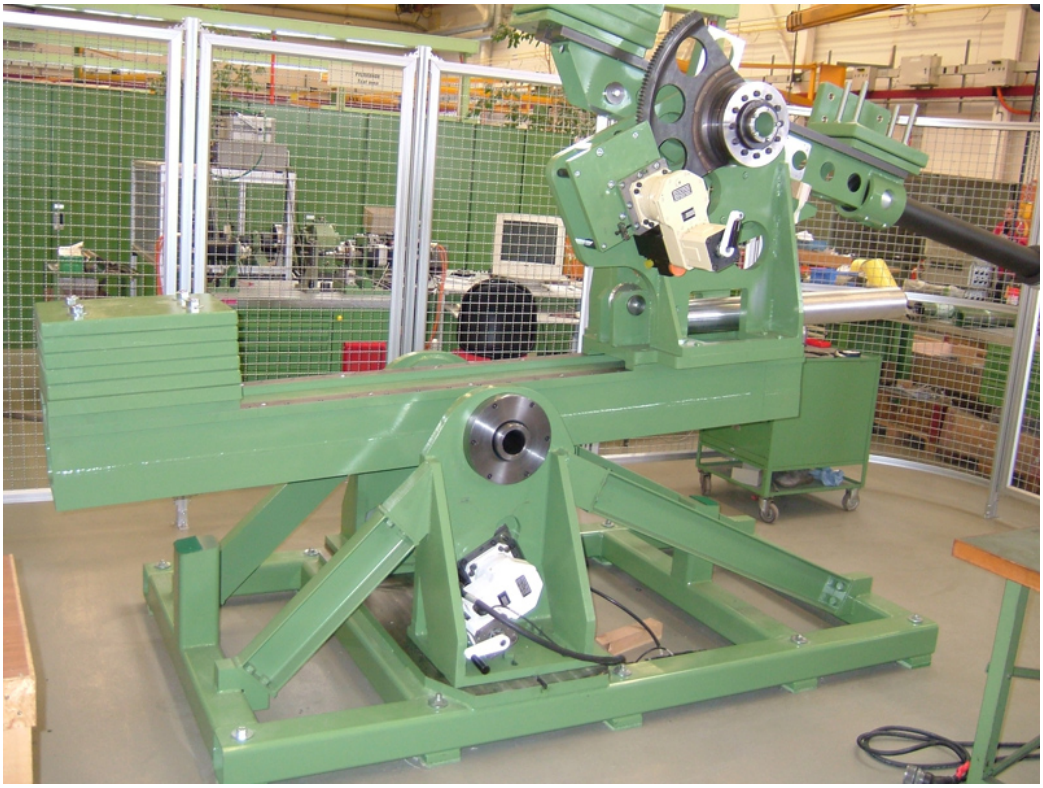


Figure 3.1: Stabilization platform

Each part are driven by one 280[A] AC/DC motor over gear segment with planet mechanisms. The vehicle ”lower part” of the platform receives it feedback from the encoder mounted back on the motor regarding its speed, position and current. The upper part which will be stabilized due to the test stands movement will receive its reference signal from a gyro. Figure 3.1 shows the stand with the motors and gearboxes. The motor and gearbox are white coloured placed low center, and up to the right.

3.2 DRUi 360

DRUi 360 is a controller unit for stabilization tasks; it is used both in military and civil industrial applications. In the military it is used for turret rotation, weapon stabilization and elevation. In civil it is used as a stabilizer by the Swiss railway in their ICN trains for the tilting control in curves. It runs with 28VDC and a maximum current of 360[A].

Figure 3.2: DRUi 360

3.3 CPU

The system CPU is a DSP from Texas instruments TMS 320LC54x which is a fixed-point DSP with a modified Harvard architecture that features low power consumption and a high degree of parallelism in the DSP core[1]. The architecture is particularly designed for real-time signal processing. The real-time capabilities make it suitable for applications that cannot tolerate delays. A DSP is a kind of specialized CPU for specific digital signal processing, which means that it is not built for so general and various purposes as a microprocessor.

3.3.1 Architecture

The modified Harvard architecture is designed with one program memory bus, three data memory buses and four address buses, total eight 16-bit

[1]TMS320C54x DSP Programmer‘s Guide

buses with on-chip memory and on-chip peripherals.

The separate program and data spaces allow simultaneous read and write of program instructions and data, which provide a higher degree of parallelism into the system[2]. It means the opportunity to read data from two memory operands at the same time over two different data buses in one single clock cycle. To read data from two single operands at the same time are under no circumstances possible with only one distinct data bus.

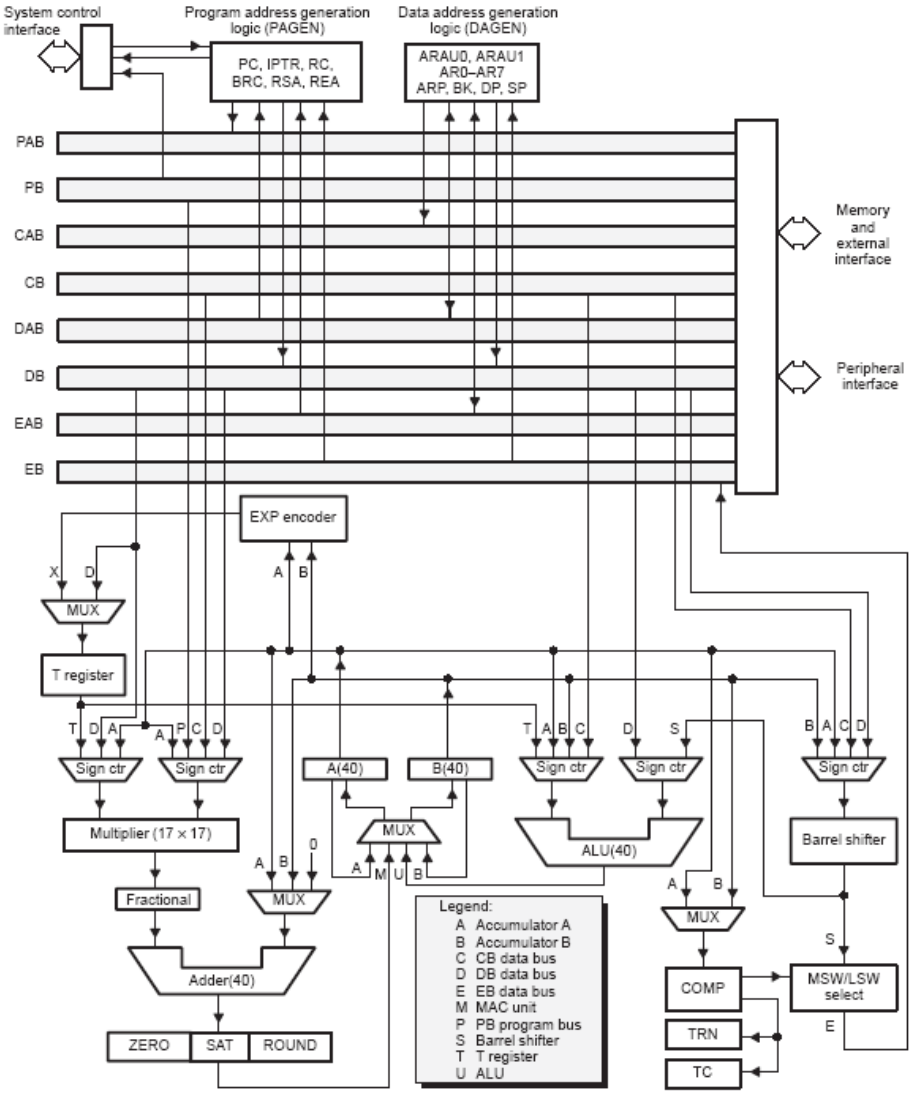


Figure 3.3: Architecture Texas Instruments c54x DSP

[2]TMS320C54x DSP CPU and Peripherals Volume 1

The c54x DSP are arranged with three 64K 16-bit memory spaces for program, data and I/O. The on-chip memories have the advantages of higher performance, lower costs and lower power consumption than an external memory. The higher performance of on-chip operations is a result from that no wait states are required, so it reaches a higher flow within the pipeline of ALU[3].

3.3.2 Features of c54x DSP

40-bit (ALU)

Two 40-bit accumulators A and B

17 *17-bit multiplier

40-bit adder

Data address generation unit

Program address generation unit

The ALU performs 2s-complement arithmetic with a 40-bit ALU and two 40-bit accumulators A and B. The ALU supports Boolean operators and use these operators.

16-bit immediate value

16-bit word from data memory

16-bit in temporary register, T

Two 16-bit words from data memory

32-bit word from data memory

40-bit word from accumulator A or B

3.3.3 Addressing modes

The c54x has seven different addressing modes, which are

Immediate addressing

Absolute addressing,

Accumulator addressing

Direct addressing

[3]TMS320C54x DSP Programmers Guide

Indirect addressing

Memory-mapped register addressing

Stack addressing

3.3.4 Serial ports

The TMS320LC548 CPU has two buffered serial ports (BSP) and one time-division multiplexed (TDM) port. The BSP is a synchronous serial port with an auto buffering unit that is clocked at full CLKOUT rate. The BSP is full-duplexed and double-buffered. The TDM port allows data to be time-division multiplexed[4]. It can be configured to handle either synchronous operations or TDM operations. The serial port is often used to communicate with other processors in a multiprocessor system for example a serial communication device, like a Universal Serial Controller from Zilog.

3.4 Zilog comunication unit

The Zilog Z16C30 USC Universal Serial Controller is a dual-channel multi protocol communications peripheral device for data communication with two independent full-duplex data channels. It could be used for serial-to-parallel or parallel-to-serial controller or converter[5]. The communication protocol is software configurable, and could therefore be used for a variety of different serial communication applications. Each receiver and transmitter channel has a FIFO buffer and it supports both asynchronous and synchronous data transfer with two baud rate generators per channel. The communication unit has the task to realise and take care of the serial communication between PC and DRUi.

[4]TMS320c54X DSP CPU and Peripherals Volume 1

[5]Zilog Z16C30 Data sheet, November 2005

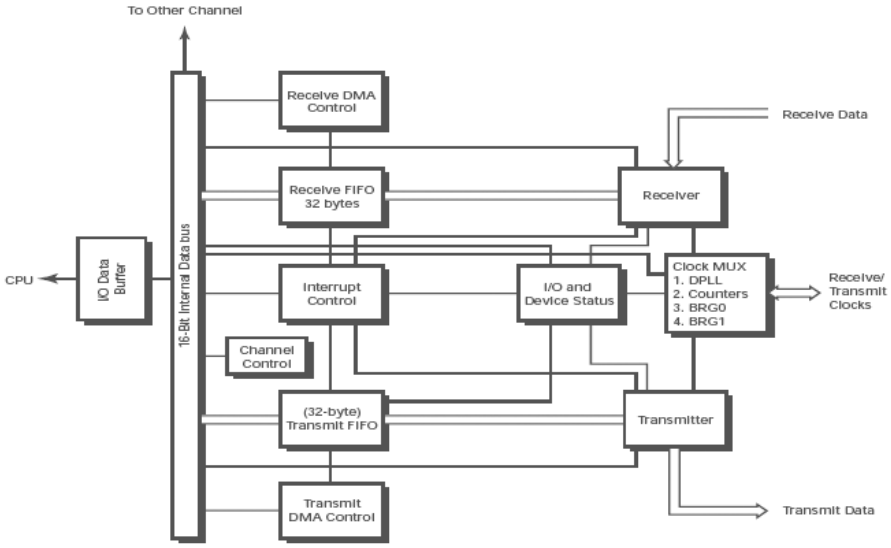


Figure 3.4: Architecture Zilog Serial Controller

3.4.1 Features of the USC

Two independent full-duplex data channels 0 - 10Mbps

32-Byte data transfer FIFO for each transmitter and receiver channel

16-Bit data bus bandwidth

110-ns bus cycle time

DMA-interface for each transmit end receive channel

The USC is the hardware interface that handles the serial communication.

3.5 USB-COMi-M

The USB-COMi-M connector is a USB device that transforms the USB-port into a serial communication port, in other words a Virtual Com port. The advantage of this serial device is the easy plug-and-play features that come with this device. It is self powered though the USB connection and the converter is easily changed between RS-232, RS-422 or RS-485 serial communication by the 4-pin DIP switches on the box. The box is connect able by one DB-9 male serial connector or one 6-pin terminal block[6].

[6]USB-COMi-M Data sheet, October 2005

Figure 3.5: USB-COMi-M

3.5.1 Features of USB-COMi-M

384 byte receive buffer.

128 byte transmit buffer.

Requires no IRQ, DMA, I/O port.

Data rates from 300bps to 921.6Kbps.

Monitor LEDs of TxD, RxD.

Power output of 5V.

3.6 XDS510 USB Emulator

For downloading the software into the CPU the XDS510 Emulator from spectrum digital are used. The Emulator is connected to the board by a JTAG connector. The Emulator supports the ability to on-board programming and testing the configuration based upon the boundary-scan (IEEE 1149.1 JTAG) technology, with the possibility to debug and diagnosticate on a system through a number of dedicated test pins[7]. The connection through the USB port delivers fast downloads and the benefits of the plug-and-play handling from windows.

3.7 AC/DC servo motor

The motor used for test and verification of the program behaviour is a 140[A] Brushless DC servo Motor (BLDC) with 28[VDC] working voltage. The mo-

[7]TMS320c54X DSP CPU and Peripherals Volume 1. Page 435

tor is synchronous and could not directly start with AC current. A synchronous servo motor has a rotor that carries a magnetic field, produced by a current excitation from a rotating rotor. As the brushless motor doesn't have any brushes that physically rotate inside the motor its movement are low sounded and no sparks appears from the rotating rotor. The BLDC motor could also be called Permanent Magnet Synchronous Motor (PMSM). The deferens between them is the back EMF, it's trapezoidal from brushless and sinusoidal from PMSM motor. The torque that are delivered from the servo motor is linear and gives maximum torque from zero speed[8].

To run the servo motors with DC current a more simple structure of the hardware and software is achieved because of fewer sensors for current control.

3.7.1 Motor feedback

The motor contains a multiturn encoder for data feedback. The encoder is both a position and speed sensor that is used for closing the speed and position loops and controlling the motor torque. The amplitude of the current is corresponding to the desired torque. The phase of the current is mounted to the position of the rotor. The encoder gives absolute information up to 4096 motor turns, with a resolution of 15bit (2^{15}) and the maximum speed of 6000rpm.

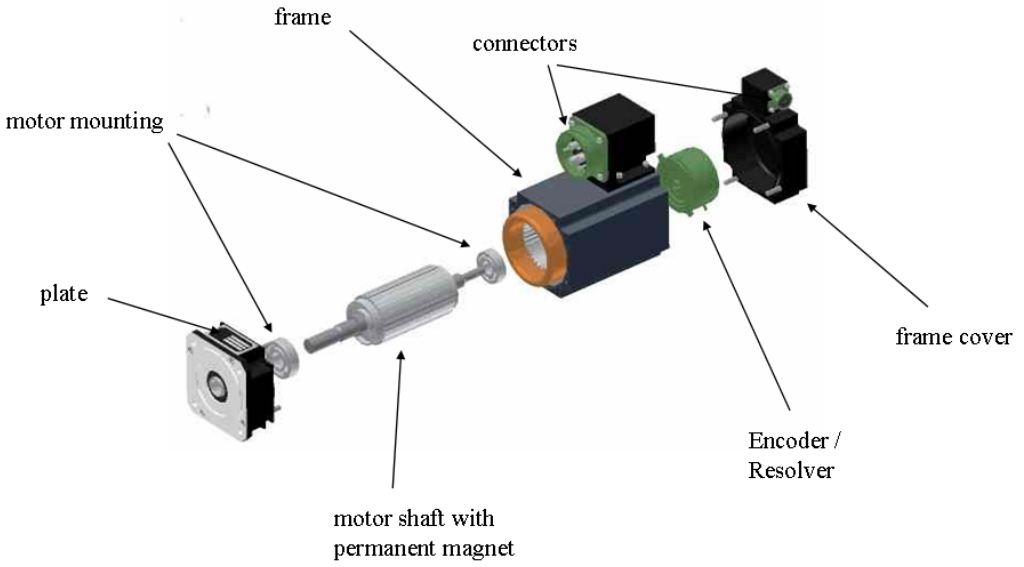


Figure 3.6: Motor parts

[8]Digital Motor Control 1-day workshop.

3.8 Softwares

The main softwares used in this project are LabVIEW 7.0 from National Instruments and Code Composer Studio 2.0 from Texas Instruments.

3.8.1 LabVIEW

LabVIEW Laboratory Virtual Instrument Engineering Workbench is a graphical programming language that is widely used for data acquisition and control systems, both in academic and industrial areas for simulation and research. LabVIEW is multi platform software that runs on various platforms as Windows, Mac OS, Linux, and Solaris. By creating a VI Virtual Instrument the PC could be used as a standard instrument with high flexibility for data acquisition[9]. Instead of writing code you normally use the standard toolboxes to create block diagrams and then compile it into machine code. LabVIEW is the software that is used to program the Ground Profile Generator. The generator was already programmed as an earlier thesis work by a student during 2002 and therefore needed some measures when it should be utilized together with the whole system.

[9]LabVIEW User Manual April 2003 Edition

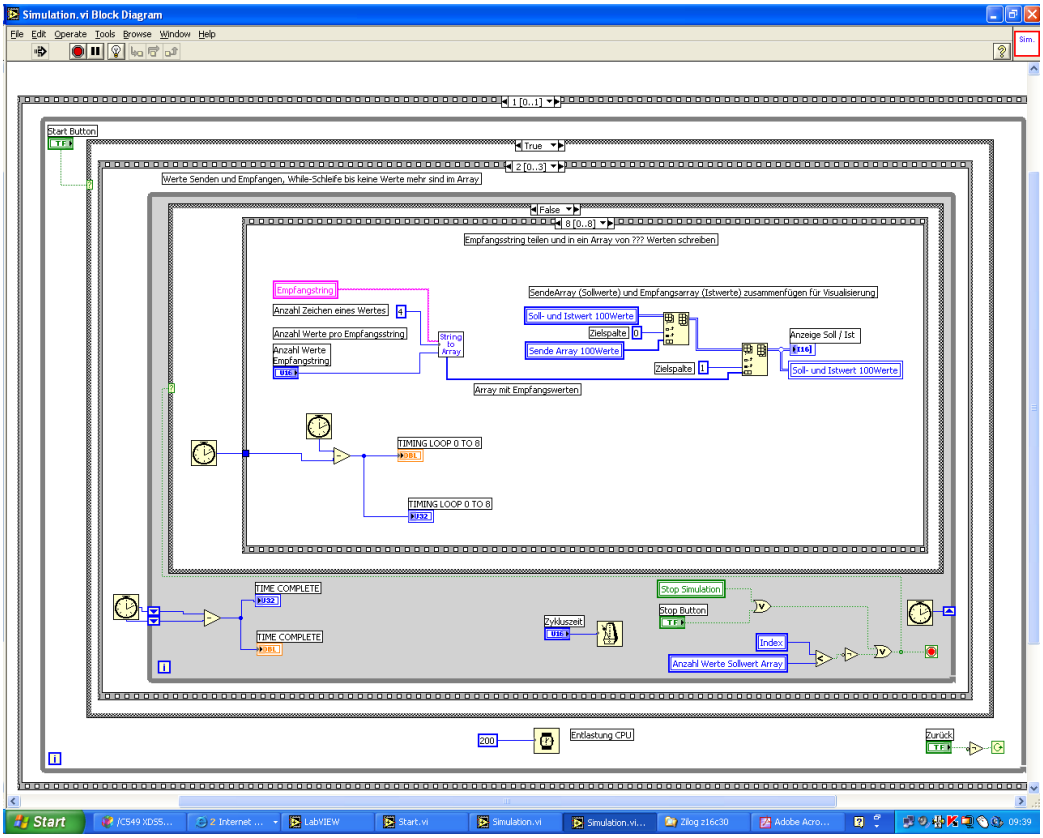


Figure 3.7: LabVIEW block diagram

3.8.2 Code Composer Studio 2.0

The Code Composer Studio is a full featured development environment to build and debug DSP software applications from Texas Instruments. Code Composer supports both C/C++ and assembler programming language. In this thesis Code Composer Studio are used as an assembler programming environment, and assembles the code into run able machine code for the target CPU. The target CPU is a Texas Instrument TMS320LC54x DSP. Code Composer Studio is an integrated development environment. It provides modern project management with editor, integrated assembler compiler and powerful debug and profiling functions[10]. In addition to the standard debug functions it is possible to visualize the information stored in RAM/Flash in graphical windows (even in frequency domain/FFT). Due to improved JTAG technology this data can be updated to these windows without stopping the

[10]Code Composer Studio IDE v2 White Paper

DSP. To download the software into the target CPU the Spectrum Digital XDS510 Emulator is used, which is described later in this report.

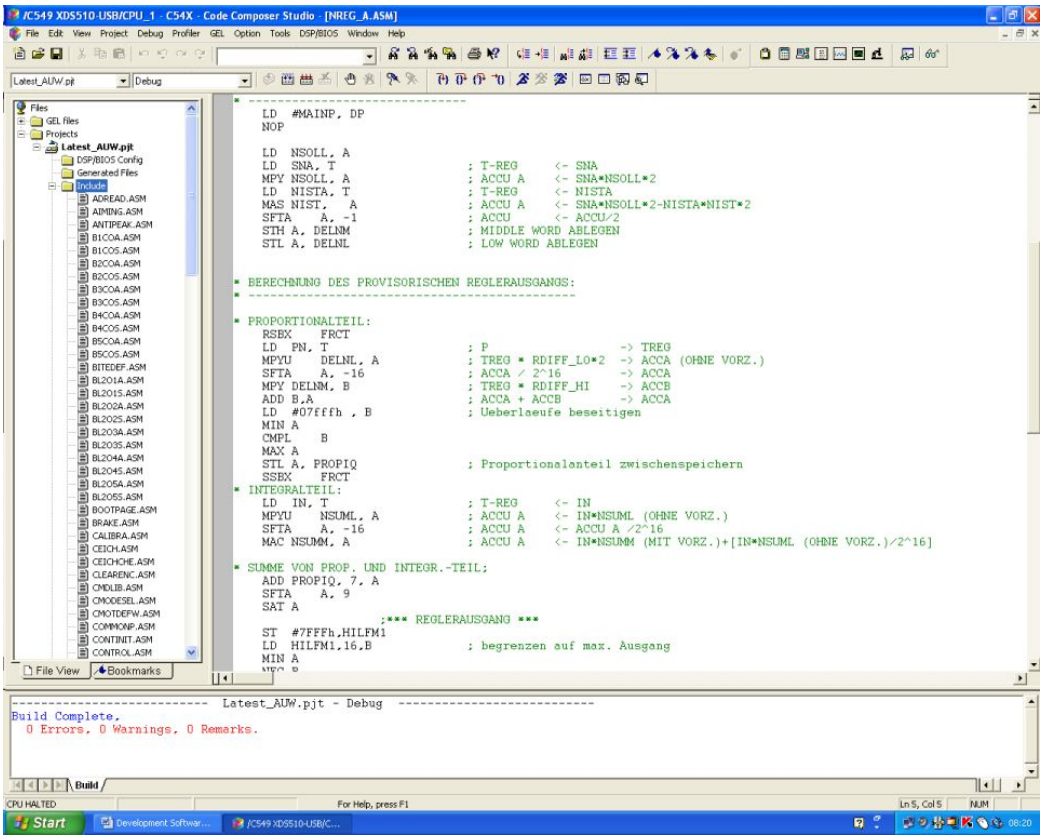


Figure 3.8: Code Composer Studio

3.8.3 Programming the DSP

The DRUi 360 is as earlier described programmed in Assembly language or more often called Assembler. Assembler is a language that requires more knowledge about the hardware when you program, compared to high level programming languages like, C++ and Java. In programming there are three major levels of programming, high level, assembler and machine code. High level could also be divided into separate groups itself. In high level programming the code could either be compiled into assembler code or directly to machine code. Machine code is a series of ones and zeros, there fore it's difficult to program and read machine code. In assembler programming you don't put together cases or structural meanings that you would do in high level programming. The assembler language also differs from which type

of processor and manufacturer that are used, but the basics and structural behaviour are almost the same. The commands in assembler are so called mnemonics, which are written first in the instruction and before all arguments[11]. An example of mnemonics for the Texas c54x DSP is a simple load instruction where the data from a variable is copied to the accumulator.

LD VARIABLE_X, A

The data in variable_x is the source and A is the destination. The old value in accumulator A is over written by the new value from the source. In assembler programming all the variables are stored either in the memory or in the processor registries. In the registries the available space are more restricted than it is in the memory. Read and write instructions through the registry is more rapid than a read and write from the memory because the internal registries at the processor.

When programming in assembler it is very important to always keep in mind the data pointer "DP" so it points on correct data page, otherwise there is a possibility that data may be overwritten in the memory pages. It is also important that the data pointer points on correct page for push and pop of the data stack and that the instructions write it values on correct page. Otherwise next instruction may not find the correct value. In high level programming like C++ you don't need to think about the data pointer and stack manipulation.

[11]Mikrodatorer bit för bit page 191.

Chapter 4

DEVELOPMENT SOFTWARE

The software in this project are adapted from other projects within the company. This chapter describes the following adaptation of the DRUi. The major part of the adaptation were already made before this project. There where weary less documentation about what was done and how it was done. The fact that the person which worked on the system before no longer worked at the company. That resulted in a difficult start of understanding the software and what the ideés of the structure the former programmer had.

4.1 Setting up the software

The software adaptation was started December 2002. And since October 2003 no further work has been done on the software. The import of the code resulted in lots of compiler error due to different versions and setting from the old software. The start pointer were not found and had to easily be re programmed with a few lines of code. All the search paths needed to manually be set again for each folder and put in right order. In the new version of Code Composer this is done when the code is moved between two different computers which makes it very easy to import and export project files. But it doesn't work 100% correct all the time.

4.2 Understanding the software

In the beginning the code were difficult to read and understand. The code contained both old and new communication, and it was not clear which were active code or not. Many functions were disabled without any comments, the

reason for these disabled functions were, the former programmer worked on debugging some parts of the system when he leaved the company. By read the code and translate them into flow diagrams it were more easy to se the dependency between different parts.

4.2.1 Flow diagram

The assembler code is like small building blocks that are put together to one piece with branches and calls between different modules. This project contains more than 130 different assembler files. Therefore it was necessary to attain a structure of what is affecting the processing and communication of data values. To get this structure more clear, flow diagrams were drawn over the different files for processing and communication of data values. By drawing these diagrams it became obvious that many files didn't affect this project, or were not affecting the actual processing of input and output data. See appendix (A) for flow diagrams of different parts of the code. The diagrams do not cover the whole part of the project; they are only presented as a rough guide line of the code structure.

4.3 Code structure

SYSCON is the main system controller. It contains the code of every function module and subroutine used for the controller. It calls all needed functions. Every software module has to jump back to the system controller module at the end of its execution.

SYSINIT is the module where all the constants and variables that are needed are initialized.

CONTINIT serves the initialization of the automatic controllers.

FINIT This module contains all initializations needed for using the digital filters.

CONTROL is a controller module that initializes important functions like AD-conversation, system mode selection (SYSMODE), temperature limiter module and other important system functions.

IN_OR_OUT is the read and write module, that handles the communication of processed and received values.

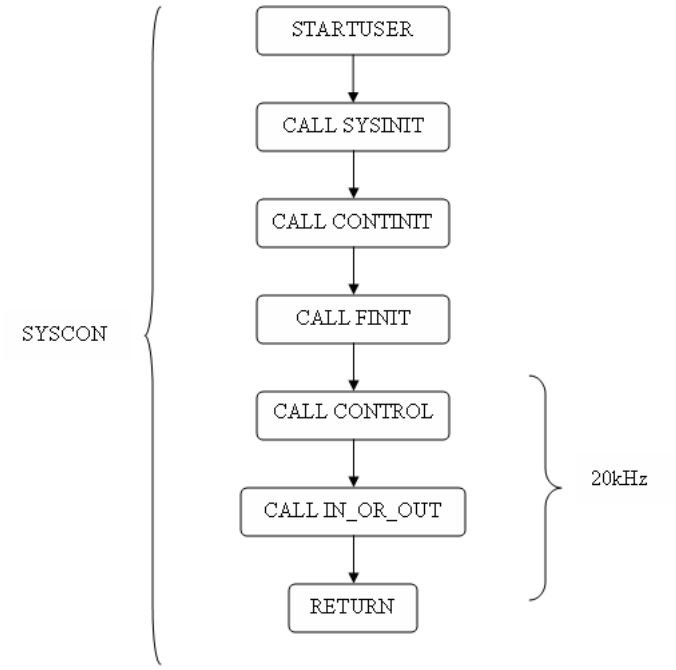


Figure 4.1: Syscon structure

CONTROL and IN_OR_OUT area runs under the 20 kHz system clock.

IN_OR_OUT are called both on positive and negative clock flank which result in a clock rate of 40 kHz. The first time IN_OR_OUT are called the bit are set to read mode "DATADIR = 0" and the input buffer are read out. When hundred values are received and processed the flag are set to one "high" and the system runs through the write mode, sends the value and shifting the memory buffers. Appendix [A] flow diagrams gives an explanation of IN_OR_OUT structure. The different system modes "SYSMODES" runs with a clock rate of 2 kHz. Possible system modes are Aiming, Standby and Exception mode. Exception mode appears only if a system error is detected. Aiming mode is the active mode for the test stand. During aiming mode the system could run either in speed mode or position mode.

4.3.1 Communication

As a first step, it was necessary to measure and assure that the communication between the PC and DRUi were correct. Earlier documentation described that 10% of the send values were not received by the DRUi. The problem was to se and know if every data value really were received and

stored in the DRUi. To se if the communication actually worked the communication were measured with an oscilloscope. The data which were send from the PC were recorded and converted in a EXCEL file and than plotted. But the result didn't really match what were expected that the PC would send. EXCEL are limited to plot fairly few values and the recording of the file didn't seem to be reliable. The solution of this problem were weary simple. Code Composer Studio has a graphical capability to present what is stored in the memory.

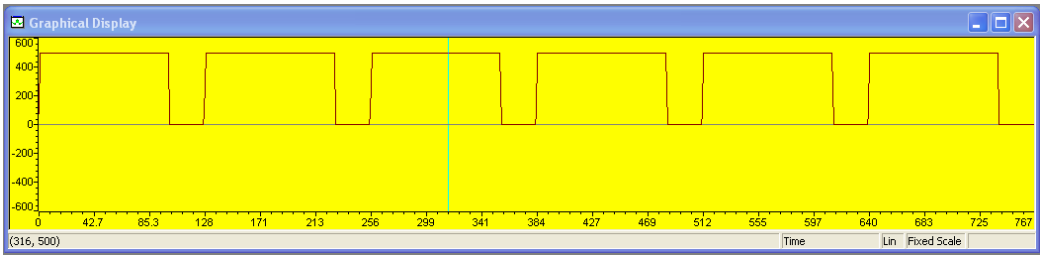


Figure 4.2: Input values

Figure 4.2 shows the circular input buffer. The buffer consists of six data pages with length 80hex which equals 128 in decimal, that's why there is a gap between every 100 input values. The conclusion were that the communication is working correct, every value are correct and no values are missing. The earlier problem of 10% loss of data values probably belonged to the older and slower computer with less recourses than todays computer.

4.3.2 Ground profile generator communication

Documentation of the project described that the problem with the communication was a problem from the Ground Profile Generator. The cause was it could not send a new data package every 200ms, the rate it reached were only a new data package every $400ms$. So effort was put on how to make the LabVIEW program sending the data with a higher rate. The generator structure is relative complex, and to do any major changes the whole structure needs to be changed. Fortunately there was a much simpler way to solve the communication issue. The processing of data values in the DRUi are easily changed by the counter VALID_VAL. It takes a new data value every $1/2000 * X$. Were 2000 is the clock frequency and X is number of cycles before next value is processed.

A new package with 100 values is sending every $400ms$. That gives the time to process a new data value every $1/2000 * 8 = 4ms$ per value and

frequency of $250Hz$. To change the LabVIEW program were not necessary for keeping the data rate of the DRUi.

4.3.3 Input and output buffer

The received values are read out in a circular ring buffer. The values are than stored in a circular buffer with six memory pages with size of 80hex each, shown in earlier picture. The processed values are stored in the output buffer. The output buffer is a switched two pages buffer page with each memory space 80hex. The reason for this switching is to have the ability to write values to one buffer and read out from the other. The input buffer starts on memory address #1400h and the output buffers start on address #1700h. Se the switching of memory pages in flow diagram appendix (A). Input file "PAGE_NSOLL_SWITCH" Output file "PAGE_SWITCH".

4.3.4 Timing received and processed values

The structure of the data processing in the DRUi is receive, process and send. Receive and processing part of code was independent and separated from each-other. To make these routines dependent, a flag are set when 100 values are received and 100 values are processed. When the statement is fulfilled the process of sending data will be initialized. The result becomes better timing and fewer disturbances between the data packets. The timing and reading of the input values are important, if some value is missing, read out to early or not processed the speed command are set to zero if there is not enough values left to process. The reason for 100 values was already chosen by the earlier programmer which worked on the system first.

4.3.5 Counting the input values

As earlier described the VALID_VAL counter are important for the number of values to be processed. Also the counter VALID_VALUES are important for the number of values that will be processed. If it not is given enough values in the initialization it will reset too early and some values will be missed in the processing. The counter is now changed by setting the processing time of every value in milliseconds instead.

$MILLISEC = 4$

$VALIDVAL = 2 * MILLISEC - 1$

$VALIDVALUES = (VALIDVAL + 1) * 100 + 2$

MILLISEC is the wanted time between every new data value to be processed. VALID_VAL is the counter for each cycle to process a new value.

VALID_VALUES are the counter for how many cycles it should count to process 100 data values in the command file. The extra + 2 are just for getting the statement jump out of its routine.

4.3.6 Processing the values

The file NIST_NSOLL contains the counters for actual and demanded values. IST "actual" and SOLL "demanded". See appendix (A) for the flow diagram of NIST_NSOLL. The demanded value and actual value are stored in the auxiliary registers for communication. The demanded value is stored in AR7 and actual value is stored in register AR6. The demanded value are than read out from its registry and than processed in the regulator and compared with the actual value. In speed mode the regulator NREG_A is used and in position mode the regulator NREG_S are used. The regulator structure is the same as in other project and therefore not needed any particular changes. The NREG_S needed only minor changes for the feedback of the position loop. Normally the file is used for stabilization with an input from Gyro. Now it gets it feedback "POSIST_HI" from the position encoder of the motor.

4.3.7 Speed and position mode

To activate the different modes the LabVIEW program need some work. The program was programmed in a previous version of LabVIEW. The statement of the command code didn't pass all the cases and therefore only worked in speed mode. In speed mode command 0022h are sending. In position mode command code 0023h are sent. To separate them a flag are set for running the system in speed or position mode. In position mode the flag closes the position loop of the DRUi and activate NREG_S which is the regulator structure for position mode.

4.4 Safety routines

For system safety additional features were implemented into the system. The safety routines are implemented to preserve the system from damages and saving the material from substantial stress. Appendix (B) contains the assembler codes of Primilim and Nlimiter.

4.4.1 Nlimiter

The Nlimiter has the purpose to prevent the system from to high accelerations. A high acceleration causes weary high tension and stress on the gear and planet mechanisms. The current could also reach very high values which increase the stress on the electronics. The limiter is a dy/dt limiter and measures the step size and step height.

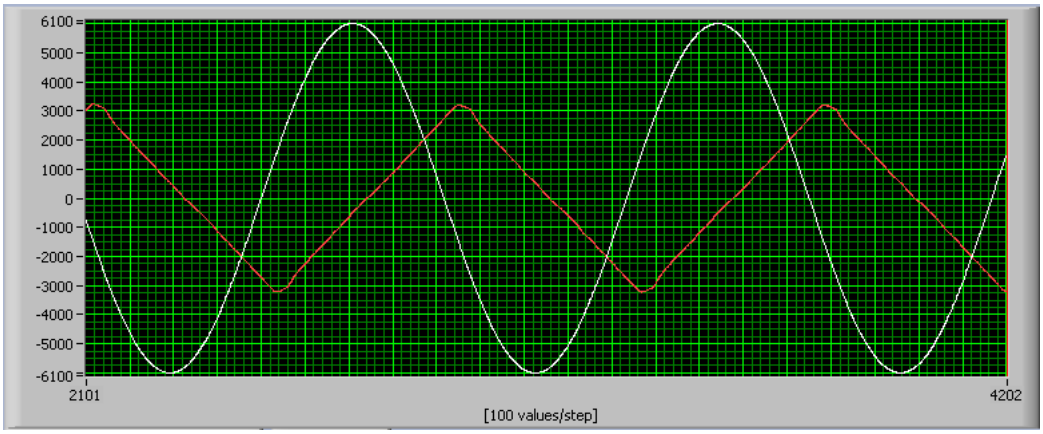


Figure 4.3: Nsoll Limiter

Figure 4.2 shows the limited NSOLL value. The input is a ordinary sinus curve. The limiter response on the input to the set maximum and minimum NSOLL value. It also save the previous demanded value for comparison with the received demanded value. Se appendix [B] for assembler code of the NLIMITER file.

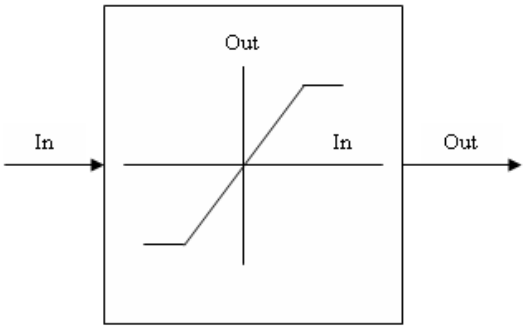


Figure 4.4: Principle of NLIMITER

4.4.2 Primilim and lower speed

The primilim is a primitive limiter that sets the speed to zero when the motor passes a given value. The primilim is useful to prevent metal to metal stops; it means that the weapon or rod not should have impact with the end stops. The given restricted value is stored in the encoder mounted on the motor. When the given limit is reached the encoder sets the speed command to zero. Also here the input signal is a simple sinus wave; the other is the actual restricted value of the motor. Se figure 4.5

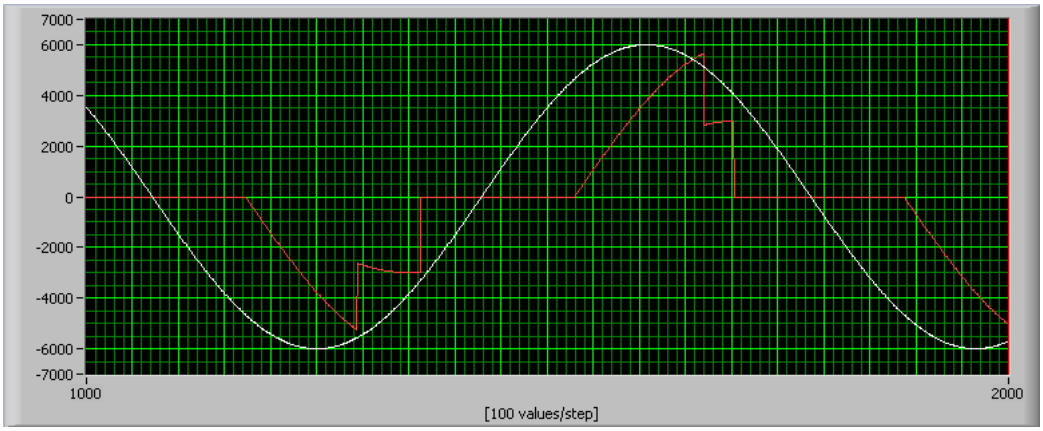


Figure 4.5: Primitiv Limiter And Limiter 50%

The limiters are implemented in two versions, one that sets the speed command to zero, and one that reduce the speed to 50% when a given limit are passed. The graph shows the speed reduction to 50% before it is set to zero speed. The reduction to 50% speed is just an extra precaution for the test stand. Figure 4.6 shows the principle of the valid areas of speed reduction. The figure are not scaled and therefore only shows the principle of the limiter.

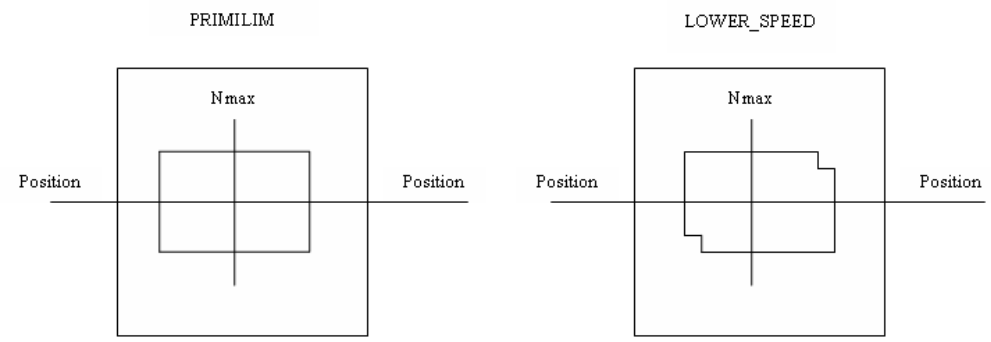


Figure 4.6: PRIMILIM and LOWER_SPEED principle

4.4.3 Aimbit and stdbybit

The system was from the beginning only set to standby mode by overwriting the command codes. To improve the control of the system the different SYSMODES had to be activated again. When the system are reset the command for standby are loaded in to the accumulator. When the DRUi receives a new data package with a valid command code, the aimbit are set and the system goes into active mode. If the communication fails and no new values are received the system goes automatically to standby mode again.

4.4.4 Stop command

The simulation menu in the ground profile generator stopped sending values when the stop button was activated and nothing more. To receive more control over the system an extra true case were added in the Ground Profile Generator. When stop button are latched the generator sends a stop code to the DRUi ”code 0024h”. The stop code then activates the standby mode and the system goes into system reset, then it is resetting the initialization values and empty the input receive and output send buffer. No old values are allowed to be in the send buffer when the system starts again. To reset the memory a loop is storing zeros to all the memory positions were speed and position commands are stored. The reason is to make the system more reliable. It could be insecure if the system has a very high input signal still stored in the output buffer when a user starts the system. The DRUi still knows the position of the motor when it is started again. The value is stored in the encoder mounted on the engine and not in the DRUi itself.

4.4.5 Brake

The motor and gearbox are connected to a handle crank. When the system are in active mode the brake bit are set and the brake are released from the handle crank and it's not possible to move the motor and gearbox by hand. When the brake bit are reset the brake are clutched and it is possible to move the rod or armoury part by hand.

4.4.6 Zurrcon

The Zurrcon is a controller to check if the drive is blocked. If the system is blocked zurrcon shuts the system down when the current reach the limit 50% and the rod still not is moving. Without this security the electronics could easily over heat and suffer from damage.

4.5 Debugging the software

This is the part of the project that has consumed most time during the whole project, together with the on going work of understanding the structure and behaviour of the software. The difficulties to debug this system appear from the cause there is no connection to a serial terminal window or similar which is very useful and common on other DSP systems. Also the lack of experience of this type of system makes the debugging difficult. But there is still some ways of debugging the system which is described further in this chapter.

4.5.1 Timing debug

As a first step of debugging the system break points were set and the system were stepped through. That give a god insight how all the counters and routines were connected together. The problem with stepping the system is that the behaviour could be different when the system runs with full clock speed. By stepping the system it was possible to see that the VALID_VALUES counter were set to low and therefore reset to early. It also caused that the output buffers contained less than 100 values.

4.5.2 Debug buffer

To get even more insight in the system two debug buffers were programmed. Both buffers are circular ring buffers that overwrite its own values when it?s full. The length of the buffer is only 400hex so the number of values is very limited in the buffer, especially when you measure counters with an

update rate of 2 kHz. To write the both input and output counter into the buffers showed that they were resetting at the same time and therefore are synchronized.

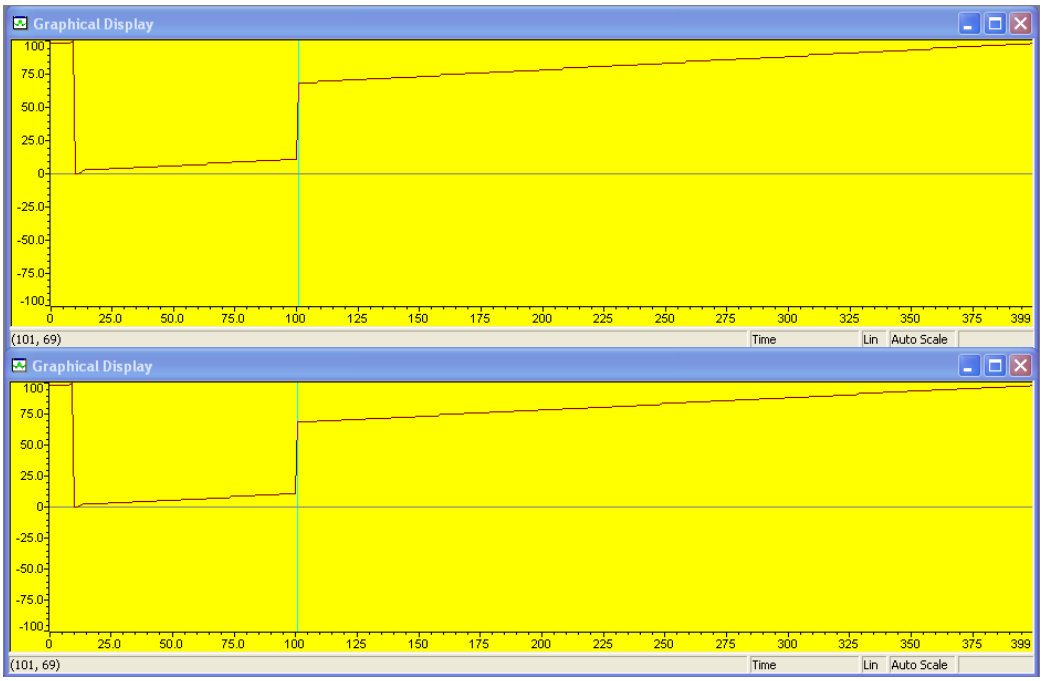


Figure 4.7: Buffer actual and demanded values

The upper graph in figure 4.7 shows the counter for actual values and the lower one shows the counter of demanded values.

4.5.3 XF-flag

Another way to debug the system is to solder a wire at pin 27 on the DSP. That pin is the external output of the XF-flag. The XF-flag is easy to measure when it goes high and low. To set the xf-flag the commando SSBX 1, XF are written, and resetting the flag with SSBX 1, XF. By setting the flag in front of a routine and reset it after the routine it's easily possible to measure the clock rate through that particularly routine or a whole file.

4.5.4 Braking the current loop

By feeding the current loop with a constant value and break the speed and position loop, is it possible to see if the problem is isolated into the current loop. By breaking the loop and write the values into the debug buffer it was

possible to see that the current loop were working correct and its regulator. By closing the speed loop and monitor the current it was obvious that the problem appears when the speed loop are closed. Therefore the problem was limited to the position and speed loop.

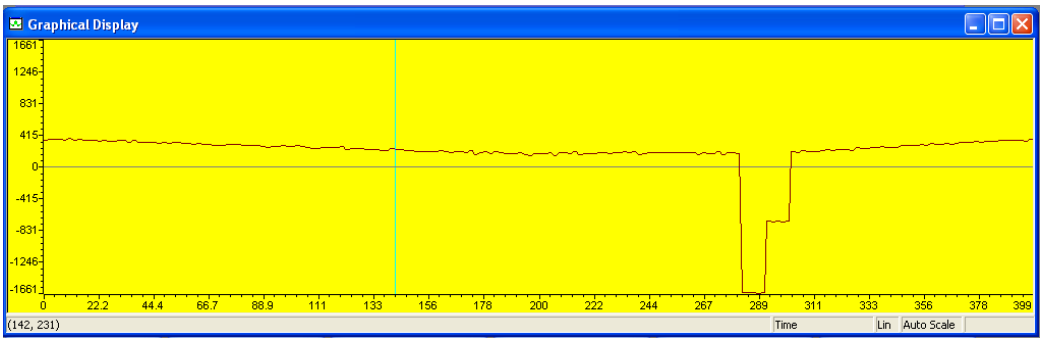


Figure 4.8: Current loop

The current loop figure 4.8 shows the disturbance when the system counters drifts away and get wrong values or missing some values.

4.5.5 Flag for last processed value

To get the gap even shorter between every send data package from the DRUi a flag were set for start processing the new values next cycle when the last value were processed. Also a flag were set for reduce the number of cycles after processing the Command code and The Lifesign. It was a mistake to do that, it actually gave a better result and reduced the disturbance, but the cause of the disturbance didn't belong from missing values or time gaps between the processing and output pages any more.

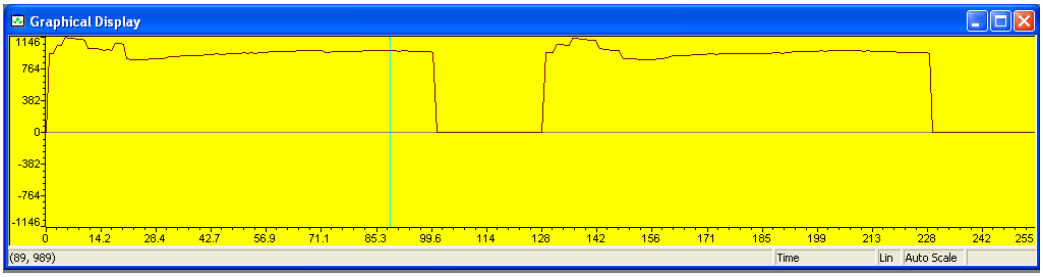


Figure 4.9: Output values with disturbance

Figure 4.9 shows that the first number of data values are correct in the output buffer, and the disturbance starts to appear later in the buffer. The

conclusion is that the processing of new data values and the switching of data pages works correct and the disturbance has nothing to do with the time gap between the processing of the data packages. So the gap between the data package are now correct. A disturbance from errors of the data values between the packages would appear from value one if it were a problem. Any regulator problem or other problem would be constant and have the same effect over the whole processed data package.

4.5.6 DRUi Sending Data

By measuring the time length of the send and received data it was obvious when counting the number of values which contains disturbance in the output buffer it corresponded some how to the time of sending out a data package from the DRUi to the PC.

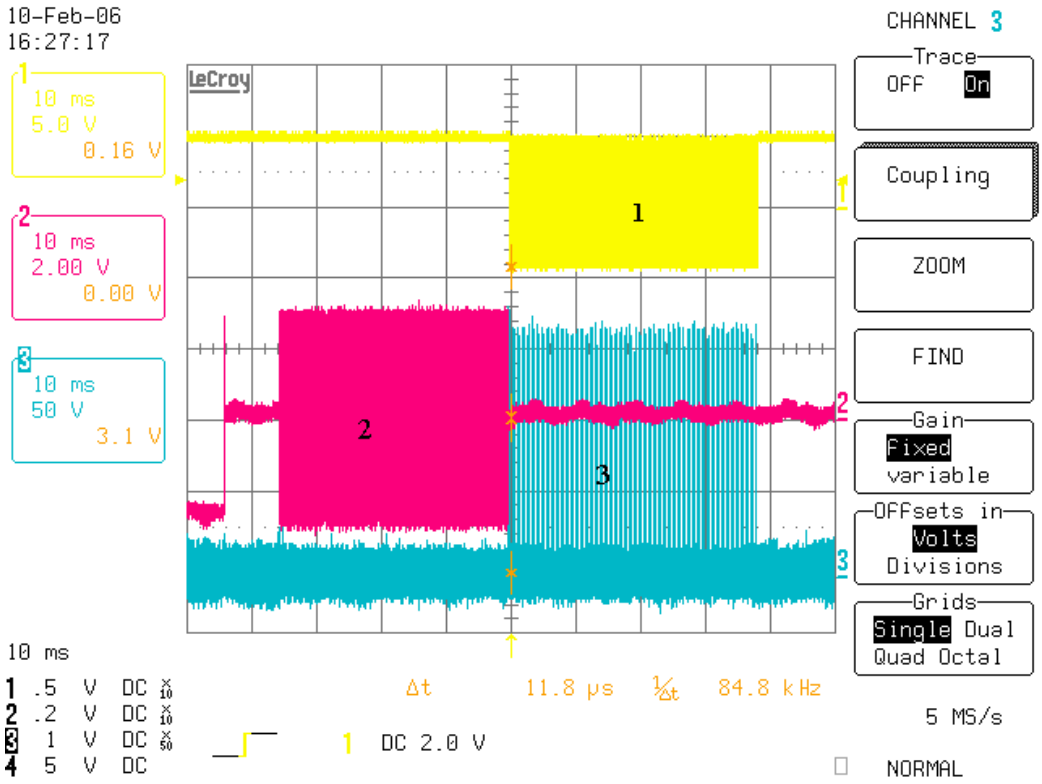


Figure 4.10: PC-DRUi Communication

1. DRUi sending data to PC
2. PC sending data to DRUi

3. XF-flag set to present the cycles for writing values out of the send buffer from DRUi.

By writing the current out into the debug buffer and monitor it during the send phase, is it obvious that the disturbance somehow has to do with the sending of data.

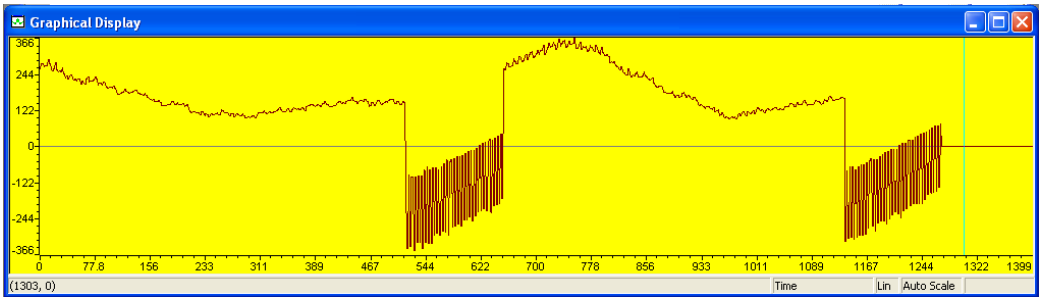


Figure 4.11: Current disturbance

4.5.7 Disabled BITF

When the disturbance at least were isolated to the send part, it was easy to follow the code and flow diagrams to the sending part. The problem in the sending part is a BITF function. The BITF function compares two values and set a flag if the statement is true or false. In this case it checks if the send port is empty or not. If not empty it will loop until the port is empty and than write out the value on the port.

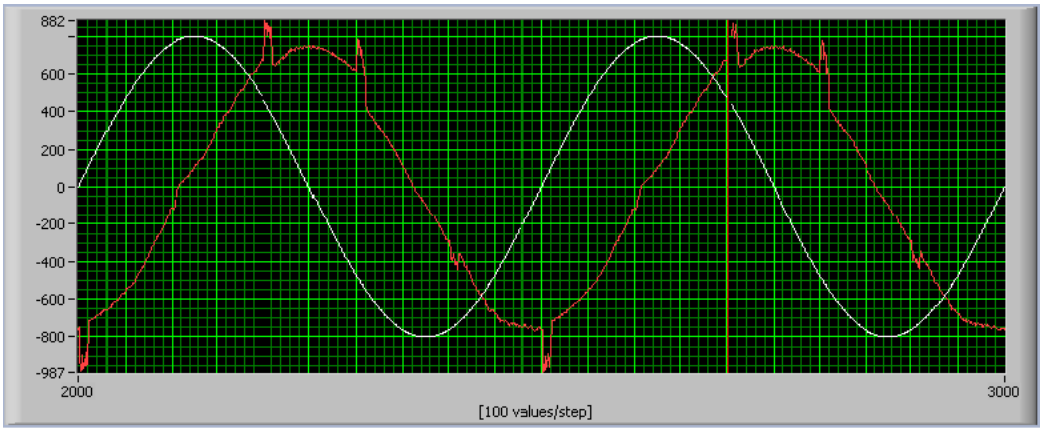


Figure 4.12: BITF active

Figure 4.12 shows clearly the disturbance when the BITF function are comparing if the send port is empty or not.

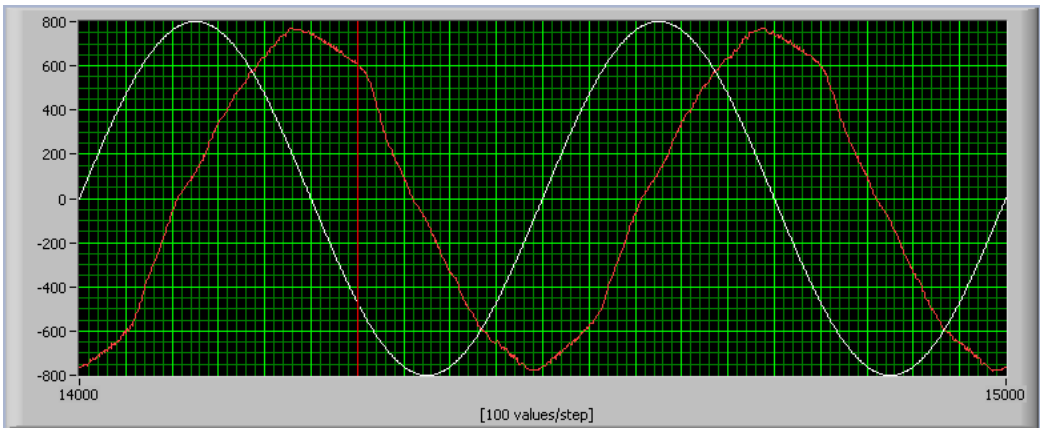


Figure 4.13: BITF disabled

Graph in figure 4.13 with the BITF function disabled. The Regulators are not proper adjusted and therefore the actual value is not that smooth.

When the BITF function was disabled the data communication working well for a couple of cycles, than other disturbances start to appear. The cause was the earlier described flags that were set after the last processed values and after the processing of the Command and Lifesign. The time gap become to short between the processing of the different data packages. By writing a temporary counter routine to give the buffer some extra time to empty itself the system are working. By doing that it's working, but there is not a good solution to have it like that. It could happen that the buffer is not empty when a value are stored in the send buffer, if there is no routine that assures that the buffer is empty before next value are stored in the send buffer.

4.6 Test stand measurements

When the system running correct and all the safety routines were implemented measurements at the test stand took part. To do the measurements another system were connected to the test stand. A same type of DRUi was used together with AXCON. AXCON is a hardware interface for communication between DRUi and PC described earlier in the report. To do the necessary measurements the AXCON has to be used. Otherwise it is not possible to measure the unbalance of the system, do a frequency analysis and

setting the right filters for the regulator structure. The PC runs the graphical interface DACSView, which is an interface for analyze and download software into AXCON and DRUi.

With the DACSView there is possible to measure all the important system parameters. It is also possible to do a FFT analyze of the closed loop and set up the filters for the drive system.

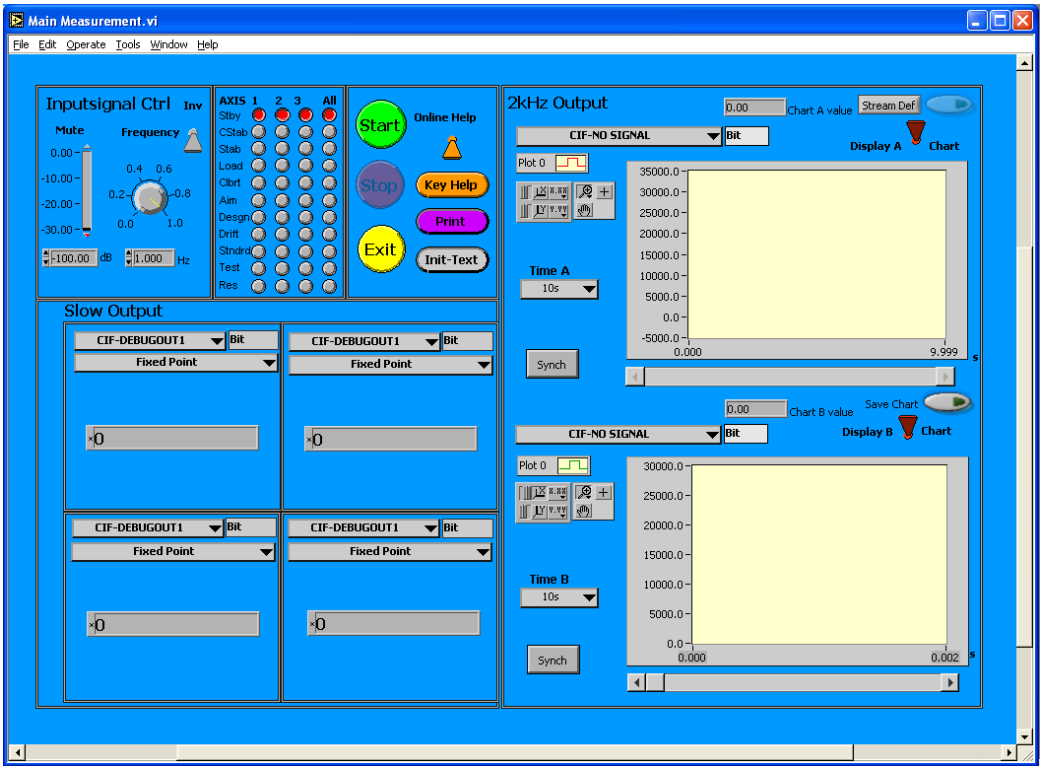


Figure 4.14: DACSView measurement menu

Figure 4.14 shows the measurement menu of the DACSView interface.

4.6.1 Teaching the system

Before running the system it has to be teached. With teaching the system means, to set up the zero position and measure the end stops to prevent the system running to forbidden areas ”end stop”. The zero point is found by setting the rod of the test stand horizontal 0° and read out the POSIST_HI value from the servo-motor encoder. The end positions are measured at the same way by turning the system by hand into the metal end stops and read out the encoder value from the servo-motor encoder. The primilim are than

set approximately one degree before the end stop.

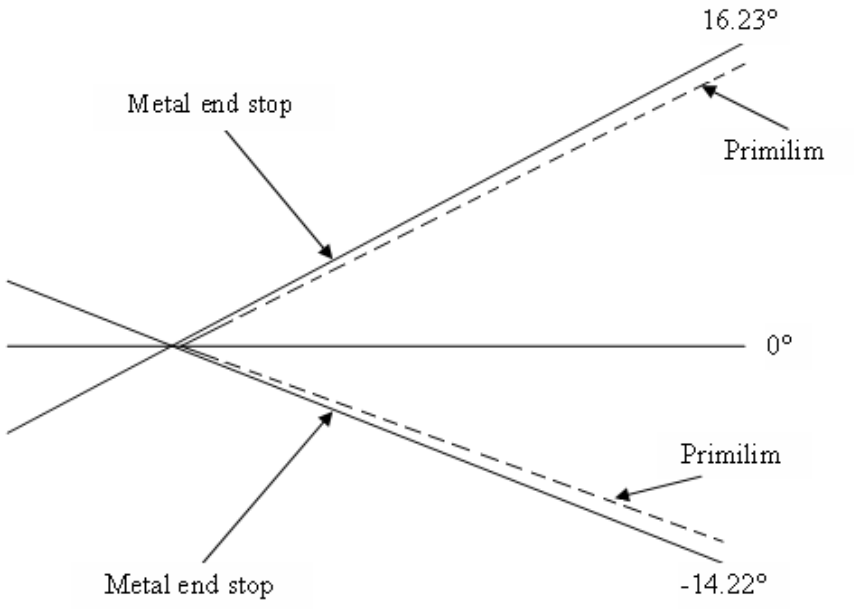


Figure 4.15: Primilim setup

Figure 4.16 shows the metal to metal stop position and the primilim setting. The angels in the figure are not scaled and therefore not actual 16.2 and -14.2 degree's.

Position	Degree	POSIST_HI	Decimal
Max	16.2°	116h	278
Zero	0°	0	0
Min	−14.2°	FF09h	-247

Through the measurements from position, and the values from the encoder there is possible to count the gear ratio and the scaling factor SNA for the speed loop feedback.

$2\pi motorrad = 16$: Encoder value/round

$\omega_{max} = 0,7 rad/s$: Maximum elevation speed

$KNISTgrob = 640$:KNISTgrob are a constant feedback scaling factor in the speed loop given by $3495/rpm$ at $50dB$

$$GearRatio = 278 * 2 * 180 * 2\pi/16.23^\circ * 16 * \pi = 385.39$$

$$SNA = GearRatio * \omega_{max} * 60 * KNISTgrob/2\pi * 6000 = 385,39 * 0.7 * 60 * 640/2\pi * 6000 \cong 275$$

All the uncommented constants are system specific constants for the particular system and has not been changed.

4.6.2 Measuring the unbalance

Before measuring and setting up the filters the unbalance of the test stand were measured.

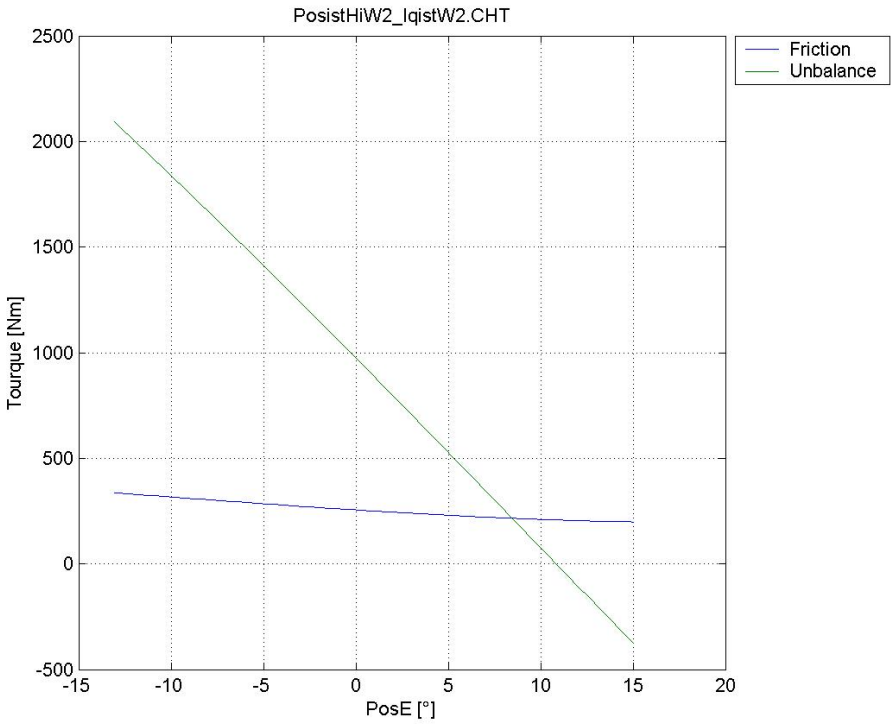


Figure 4.16: Unbalance

The measurement shows the test stands large unbalance due to its heavy front. Figure 4.16 shows the torque needed to raise the platform from negative angle to positive angle. In the negative angle "front down" the torque needed to raise the system are over $2000Nm$. In positive angle the torque are negative instead. A high unbalance causes greater stress on the material and requires more careful filter design for the accuracy. With this type of

unbalance it will be a challenge for even an experienced engineer to find a good controller/filter setup. The filter has the purpose to reduce disturbances in which appear in curtain frequency areas. The X-axis shows the elevation in degrees and Y-axis shows the torque to keep the position in zero speed. The friction of the rod is almost the same over the interval. The torque to move the rod shows that the test stand has most weight in the front and need higher current to produce more torque in negative elevation positions. To reduce the unbalance it's possible to move back the tower of the stand to equals the weight in front and back.

4.6.3 Setting up the filters

To set up the filters for the control loops a frequency response measurement were done. With DACSView the system cutoff frequency, the system resonance and filter characteristics can be determined.

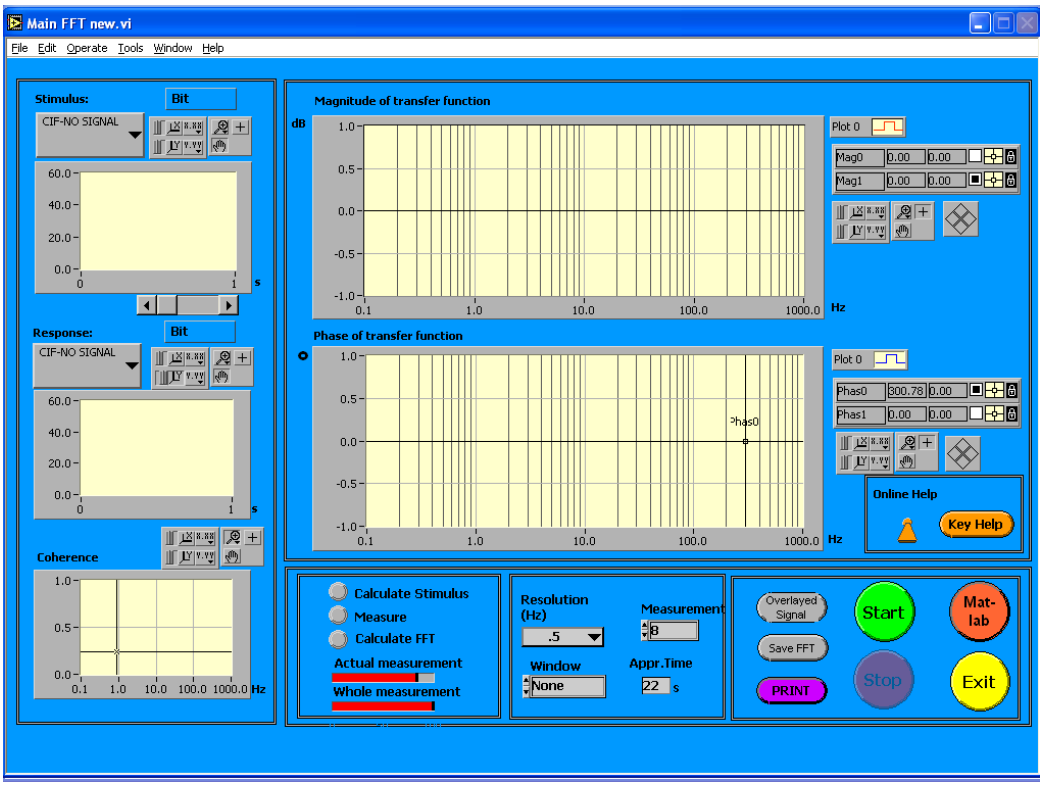


Figure 4.17: FFT window

The theory behind the filter and system optimization are complex and automatically calculated from the program. In the graphical interface, there

is possible to view the input stimulus, system response, coherence, the magnitude (dB) of the transfer function and the phase (degree) of the transfer function. After the measurement the computation of the FFT "Fast Fourier Transform" analysis are presented in a Bode plot at the center of the screen. The computation of the filter are done in a Matlab menu, which is started from the Matlab button seen in fig 4.17. The result of the measurement around 3Hz and a phase shift of over 100dB are rejected because of an error in the measurement and not a part of the filter design seen in figure 4.18.

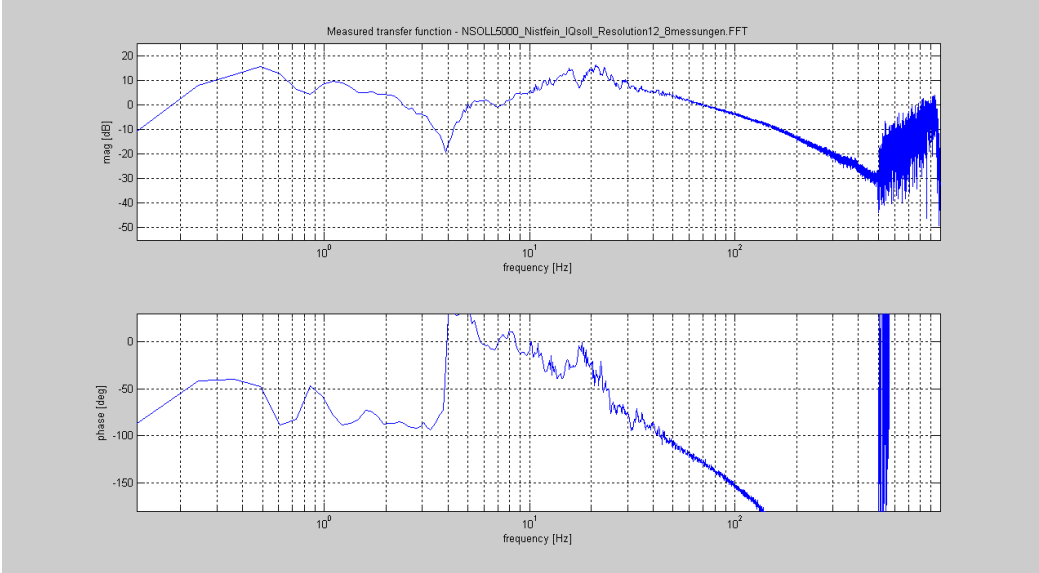


Figure 4.18: FFT Measurement

Figure 4.18 presents the FFT analysis in a bode plot diagram. The scaling of the logarithmic frequency axis is synchronized for the amplitude, phase and coherence.

4.6.4 Filters

The filters designed for the system is two notch filters. A Notch filter is a filter that passes all frequencies except those in a stop band centred on a center frequency[1]. The phase response of the notch filter has its greatest rate of change at the center frequency. In this case with the disturbances and the oscillating behaviour around 10-40Hz a damping filter is recommendable. That?s why a Notch filters with it characteristic damping factor inn the centre frequency a suitable.

[1]What is a Notch filter? January 2006

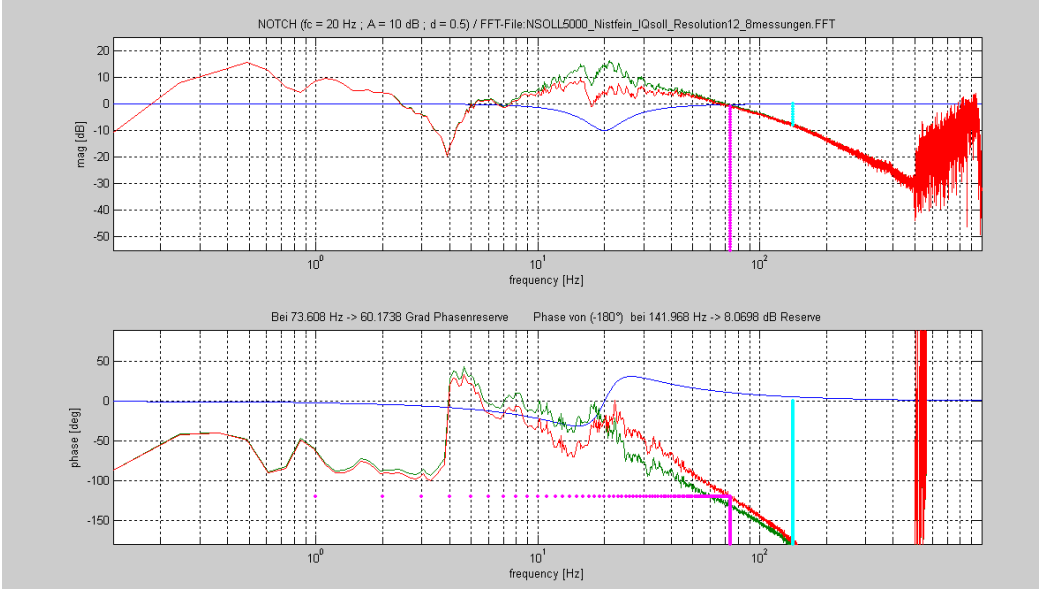

NOTCH (fc = 20 Hz ; A = 10 dB ; d = 0.5) / FFT-File:NSOLL5000_Nistfein_IQsoll_Resolution12_8messungen.FFT
mag [dB]
frequency [Hz]
Bei 73.608 Hz -> 60.1738 Grad Phasenreserve
Phase von (-180°) bei 141.968 Hz -> 8.0698 dB Reserve
phase [deg]
frequency [Hz]

Figure 4.19: Notch filter 1

In figure 4.19 the center point of the notch filter are placed at 20Hz with an amplitude of 10dB and damping of 0.5.

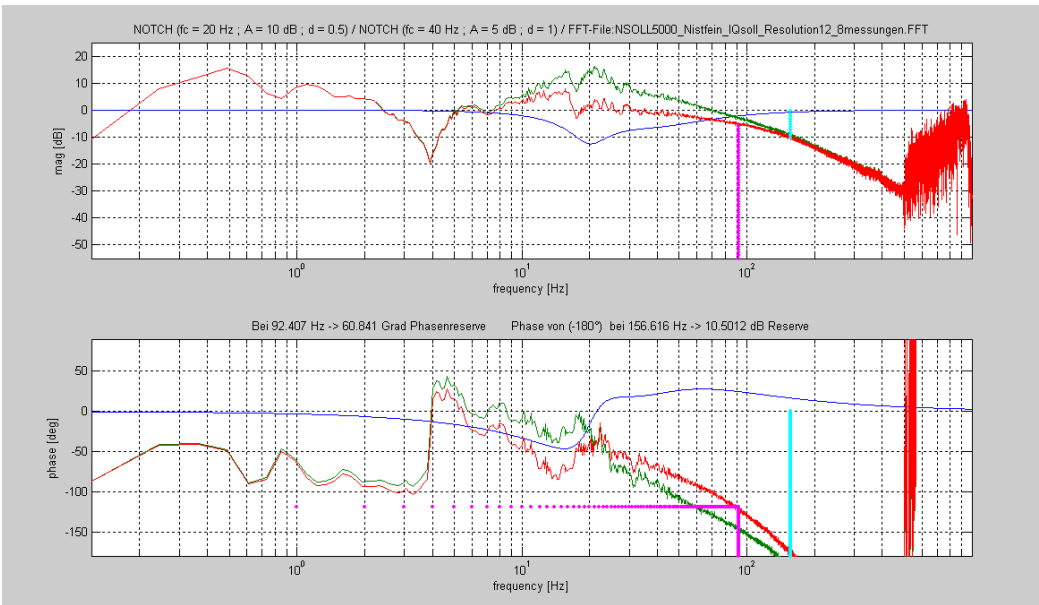

NOTCH (fc = 20 Hz ; A = 10 dB ; d = 0.5) / NOTCH (fc = 40 Hz ; A = 5 dB ; d = 1) / FFT-File:NSOLL5000_Nistfein_IQsoll_Resolution12_8messungen.FFT
mag [dB]
frequency [Hz]
Bei 92.407 Hz -> 60.841 Grad Phasenreserve
Phase von (-180°) bei 156.616 Hz -> 10.5012 dB Reserve
phase [deg]
frequency [Hz]

Figure 4.20: Notch filter 2

In figure 4.20 the center point of notch filter 2 are set at 40Hz with an

amplitude of 5dB and a damping of 1. The filter settings and regulator settings are then imported to the software developed for the test stand.

4.7 Running the test stand

When the measured parameters were imported to the new software, the ground profile generator and all needed software were installed on a Laptop for running the system at the test stand. To connect the system new power supply cables were mounted from the power supply in the workshop.

4.7.1 New boot software

To let the system run without the emulator connected to the system, new booth software with auto start are stored in the DRUi memory. The finalized software are converted through the Code Composer Studio into an ASCII file and downloaded with DACSView over AXCON to the DRUi. The reason for running without the emulator is the instability of the system with the emulator connected.

4.7.2 Disturbance

The new software has a different behavior down at the test stand. With the test system on the engineering department no disturbances appears when the system is running. The design of the filters seems to not have the same effect in the new software as the reference system were the actual measurements was done.

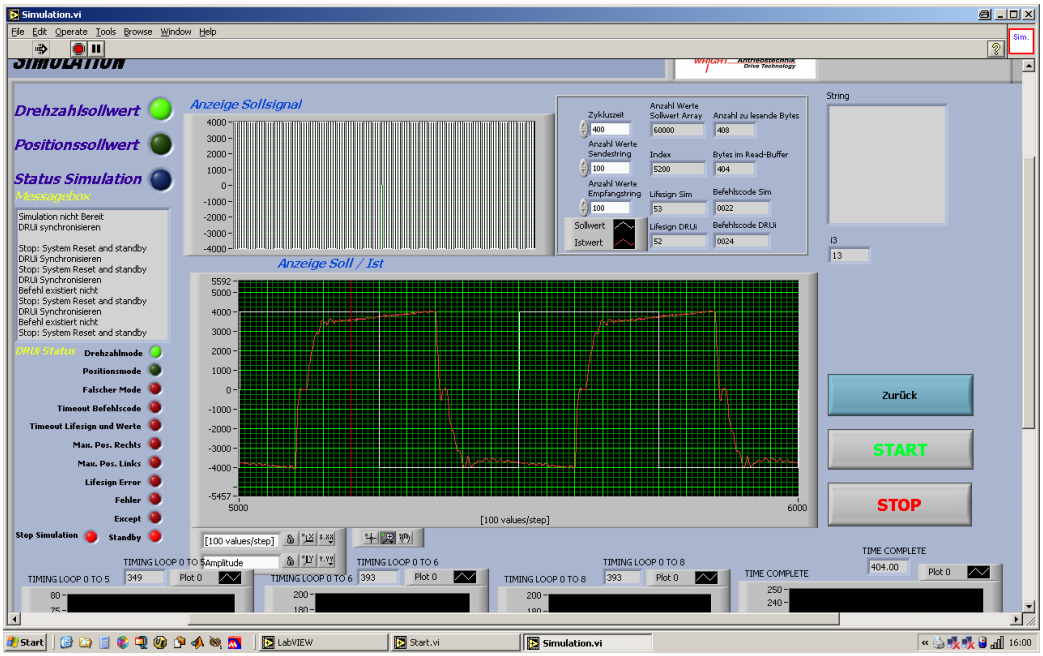


Figure 4.21: Square wave input

The simulation windows show the square wave input signal. The square wave signal is used to act as a sort of a step response signal. The result shows clearly that the regulators not are working proper, and a heavy disturbance appears around zero.

Different disturbance

The system has different disturbances in different positions.

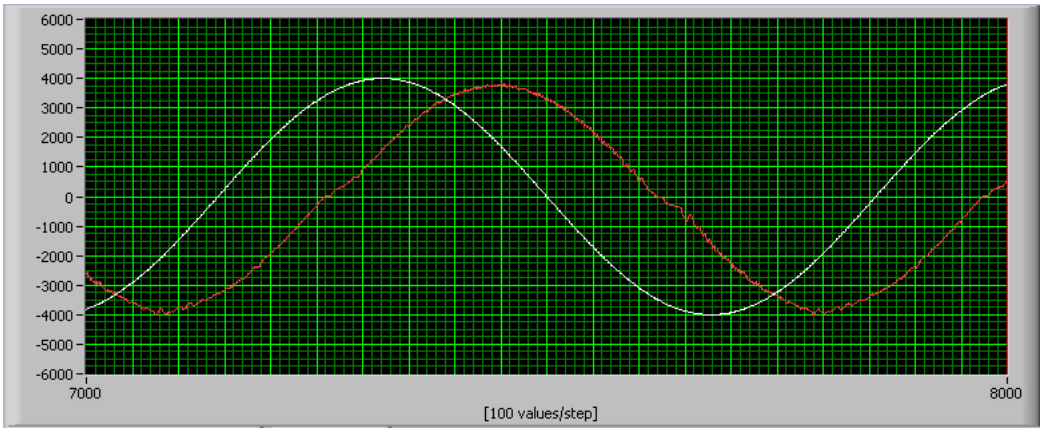


Figure 4.22: Sinus input

figure 4.22 shows the system running with a sinus input. The signal with the peak to the left are the input signal

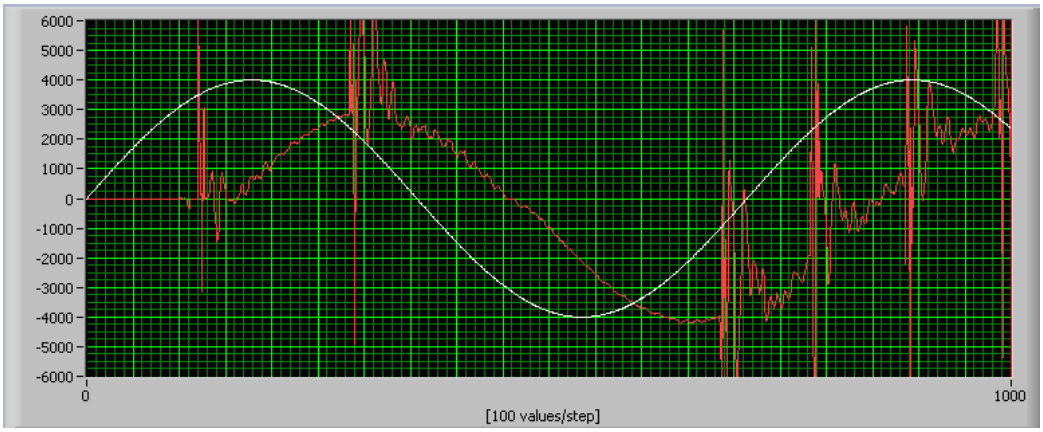


Figure 4.23: Sinus with disturbance

By manually moving the test stand to another start position these disturbances in figure 4.23 appears in the system. Moving the system back to its original start position makes it running again without any disturbances.

4.8 Debugging at the test stand

Debugging the system connected to the test stand appeared to be very difficult. Breaking the current loop or the speed loop down at the test stand is dangerous. The system falls to its end position uncontrolled and with full speed, if the user not is precautious with the values that is send in to the system. The solution is to disconnect everything and take the system back to the engineering department and proceed the debugging with only the DC-servo-motor connected to the control system. Connected to the DC-servo motor the system running without any disturbance and braking the current loop and monitor the current do not show any failure in the system.

Chapter 5

Results

5.1 Conclusions and discussion

Not all the requirements are achieved at the end of this project. The test stand is not running that smooth as it ought to do. The system runs correct in the development environment but do not work satisfactory mounted together with the test stand. During the project the emulator which is used to download the compiled software into the DRUi has malfunctioned. The new code is not always proper downloaded into the software and therefore the debugging of the system has taken long time to accomplish. The changes in the program code hasn't always affected the system, and correct been transferred to the system by the emulator. By braking the power and reset the system the behaviour was sometimes totally different than before

When all the disturbances have been corrected at the system caused by the earlier timing problems new errors has been found in the system. The disturbance which appears in the system when DRUi sending its data to the PC is not solved yet. The temporary solution of comment out the comparing BITF function from the software is not engineering manner. It should be there to assure that the send buffer is empty before a new value is placed at the port.

When the system is connected together with the test stand new disturbances appears in the system again. There could be many reasons for these disturbances. Down at the test stand much higher current and torque are affecting the system. With high currents there is possible to receive inductive disturbances in the signal cables. If there is a communication problem many different errors could appear in the system. There is a possibility that underflow or overflow of data appear some where in the system.

All the important security functions are implemented in the system and

working correct. The different system modes are also working and the system switches from standby to active mode on given speed, position commands from the Ground Profile Generator. Also the Brake is working so the system is possible to move with the handle crank in standby mode, not in active mode. The system is easy to use and implement for further test together with the test stand. Also the Ground Profile Generator has been minor improved from the previous version.

The code was not documented by the earlier programmer, so the start and proceeding in the beginning of the project were very difficult. That experience shows how important proper documentation is for the next person who should proceed with a project. To proceed on an already started project or take over a project from another engineer is a big part of engineering and therefore it has been a very god experience.

5.2 Further work

Still many improvements are possible to do on the system. On the Ground Profile Generator further work could be accomplished by synchronize the actual value with the demanded value to prevent the shifting by 100 values in the graphical window. The routine for sending the stop code are not always working satisfied, the stop code are not send and the user interface hangs up sometimes. When shifting between the different menus windows the system freezes for a short period. The rate of sending data package could further be investigated to achieve a higher data rate.

On the DRUi software there is still a minor problem with the upward communication in the WRITE function. There is this disturbance with the BITF function that needs further investigation. The input values of the DRUi could increase, instead of filling up the buffer, and process the values. The input memory buffer could be filled with two or even three data packages before it start to process and sending data back to the PC. To achieve that the Ground profile generators start sequence must be changed, so it's not dependent of sending new data when it receives a data package from the DRUi. An alternative to that is to let the DRUi send packages with zeros as it does in the first sequence for two or three data package cycles.

References

[**1**] Digital Motor Control 1-day workshop Student Guide. Revision 4.0 February 2004 Texas Instruments.

[**2**] LabVIEW User Manual April 2003 Edition Part number 320999E-01

[**3**] Mikrodatorer bit för bit. Rune Körnefors student litteratur Lund 1996 2:a upplagan ISBN 9144308620

[**4**] Motion Control Handbook by Dr. Stephen J. O´Neil Vice Precident Advanced Research and Planning. Micro Mo Electronics, Inc micromo `www.micromo.com`

[**5**] Rs 422 and 485 Standars Overview and System Configurations. Texas Instruments Application Report SLLA070B June 2001.

[**6**] TMS320c54X DSP CPU and Peripherals Volume 1 Literature Number: SPRU13F April 1999

[**7**] TMS320c54X DSP Mnemonic Instruction Set Volume 2 Literature Number: SPRU172C March 2001

[**8**] TMS320c54X DSP Programmer's Guide Literature Number: SPRU538 July 2001

[**9**] Code Composer Studio IDE v2 White Acsessed Paper November, 2005 `http://focus.ti.com/lit/an/spra004/spra004.pdf`

[**10**] Proportional and Integral Controllers, January 2006 `http://www.ele.auckland.ac.nz/~covic/sctrl_db.PDF`

[**11**] USB-COMi-M Datasheet, October 2005 `http://www.coolgear.com/USBGEAR-USB-COMi-M/Datasheet.pdf`

[**12**] What is a Notch Filter? January 2006 `http://www-k.ext.ti.com/SRVS/Data/ti/KnowledgeBases/analog/document/faqs/notch.htm`

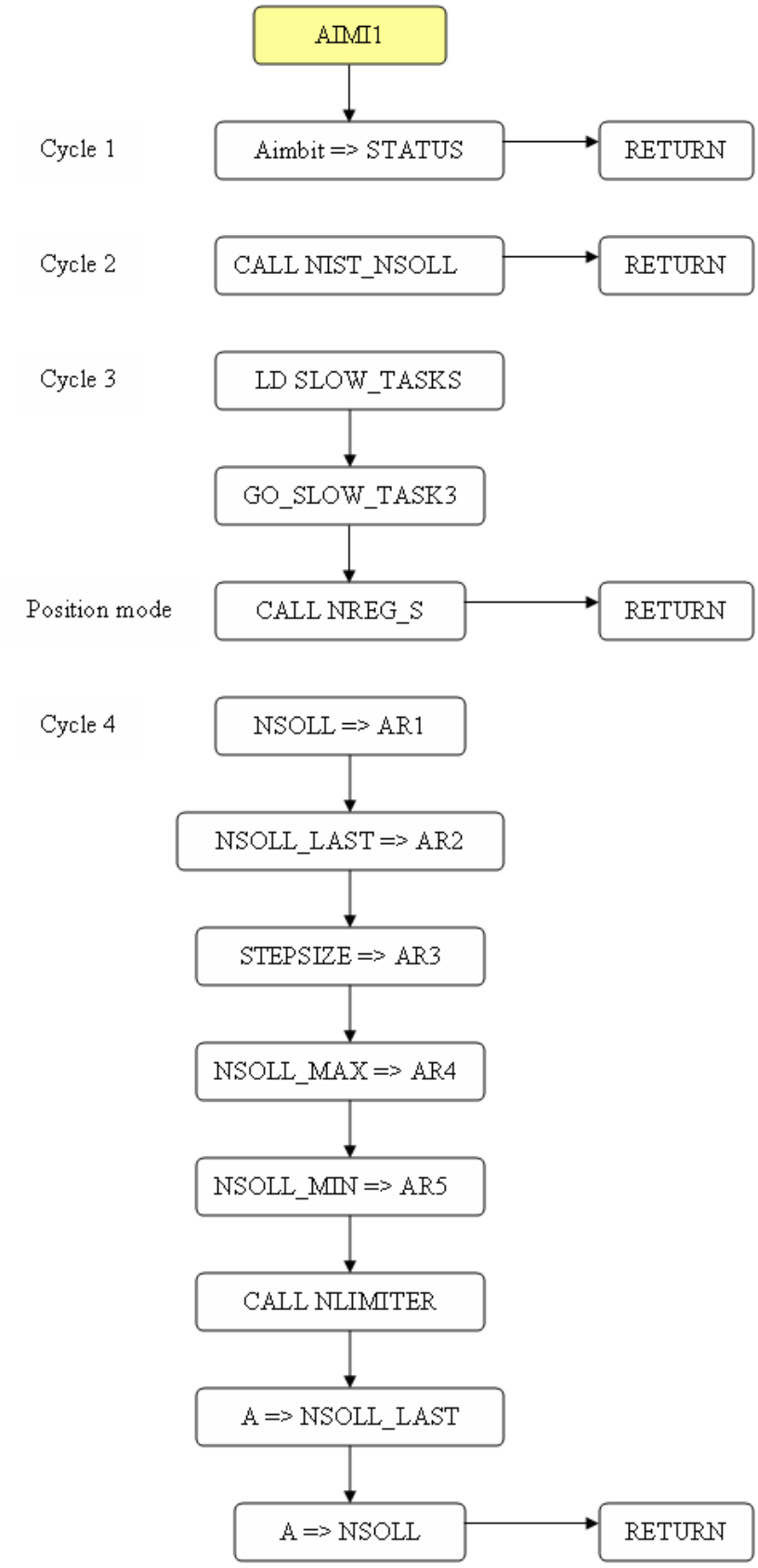
[**13**] Zilog Z16C30 Datasheet, November 2005 `http://www.zilog.com/docs/serial/z16c30.pdf`

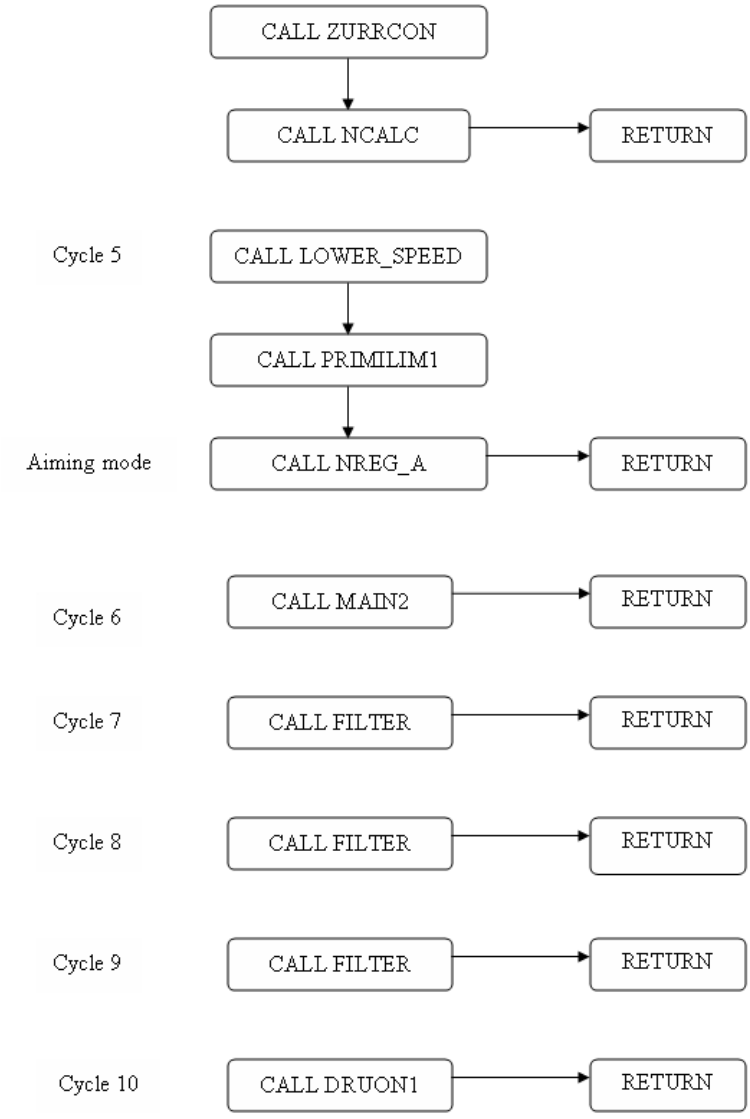
Appendix A

Code flow diagram

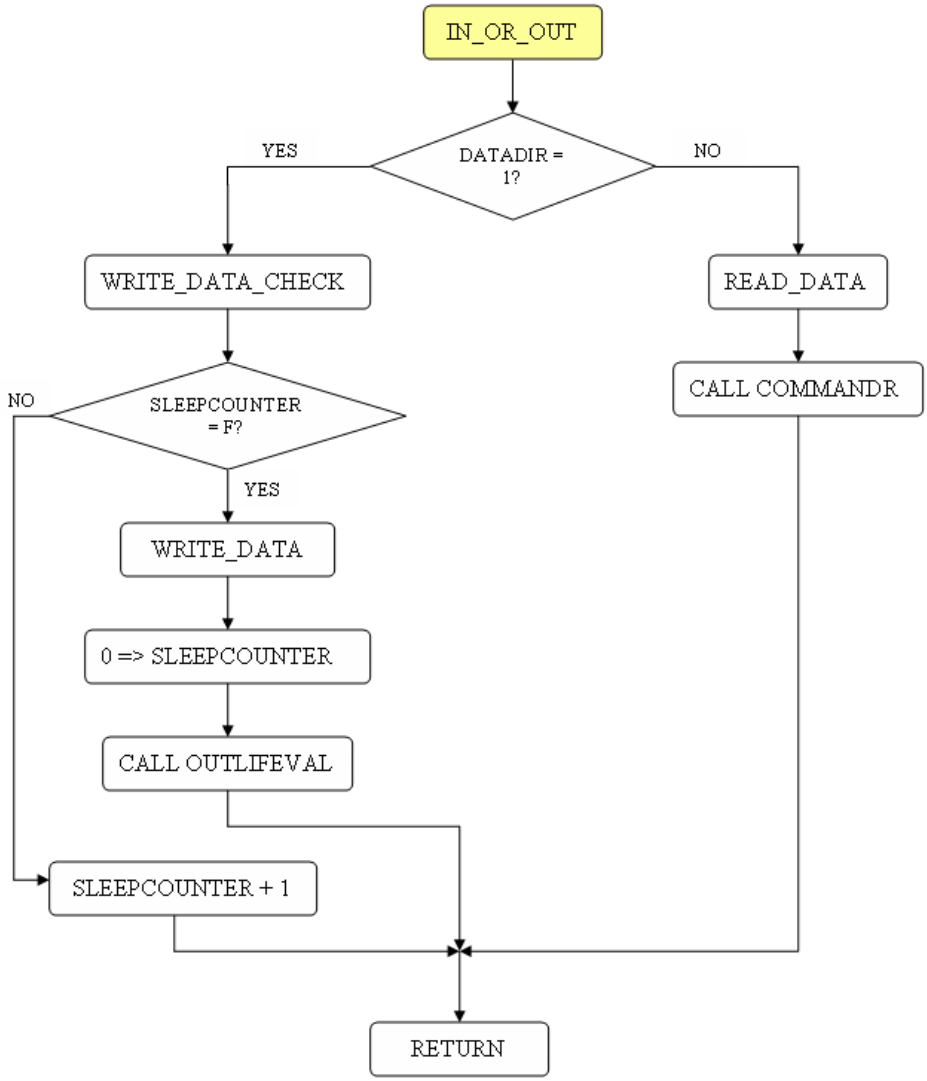
In this appendix the flow diagrams over the code structure are presented. The flow diagrams do not cover the whole code structure; it is just a part and should be considered as a guide line for how the code is structured.

AIMING

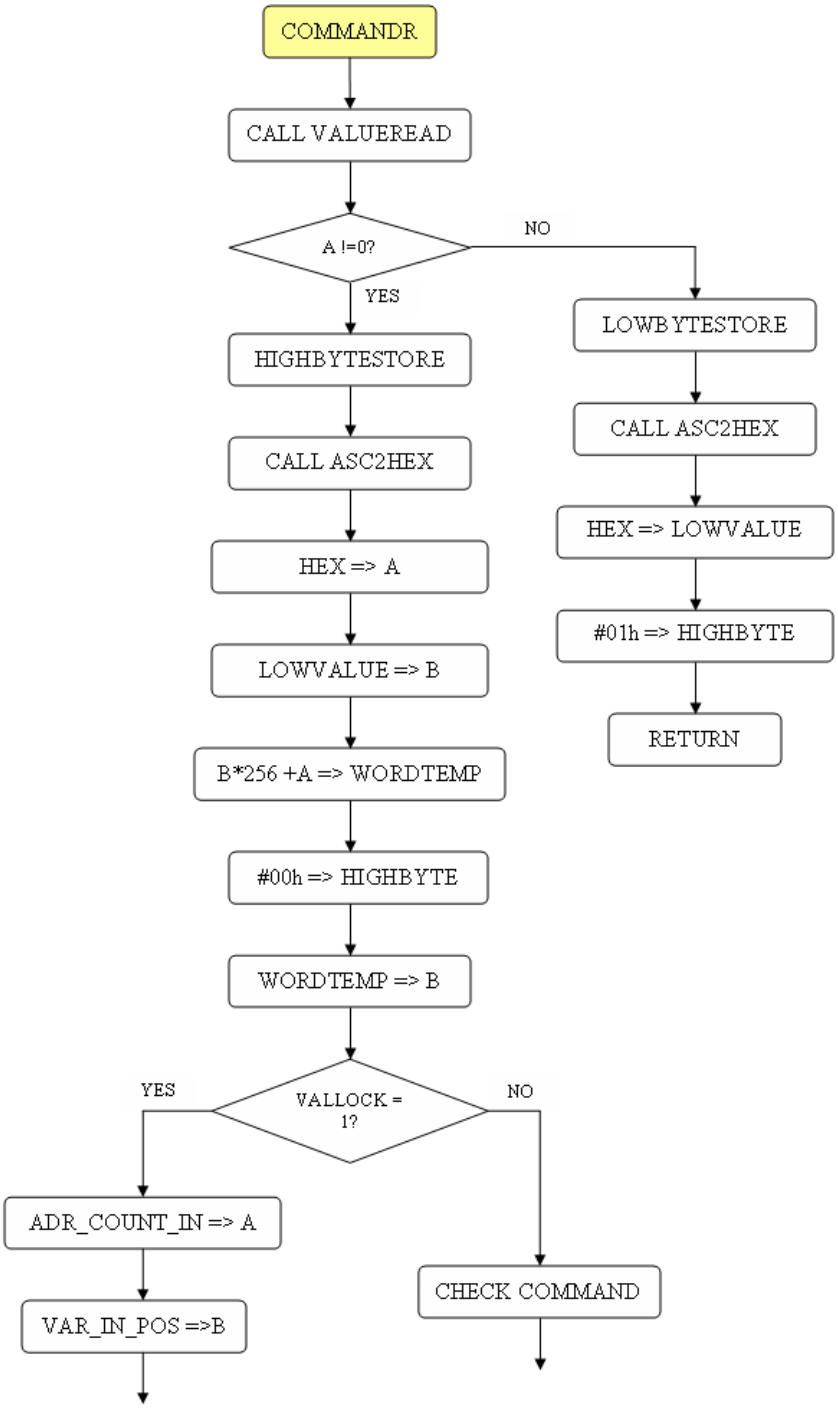

AIMI1
Cycle 1
Aimbit => STATUS
RETURN
Cycle 2
CALL NIST_NSOLL
RETURN
Cycle 3
LD SLOW_TASKS
GO_SLOW_TASK3
Position mode
CALL NREG_S
RETURN
Cycle 4
NSOLL => AR1
NSOLL_LAST => AR2
STEPSIZE => AR3
NSOLL_MAX => AR4
NSOLL_MIN => AR5
CALL NLIMITER
A => NSOLL_LAST
A => NSOLL
RETURN

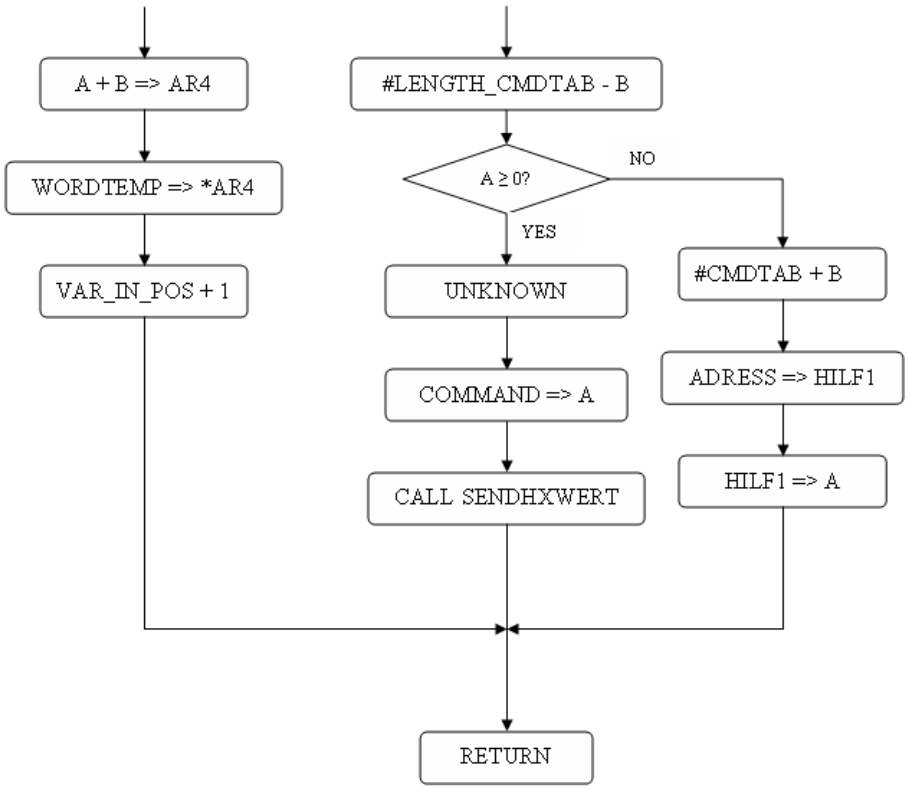

CALL ZURRCON
CALL NCALC
RETURN
Cycle 5
CALL LOWER_SPEED
CALL PRIMILIM1
Aiming mode
CALL NREG_A
RETURN
Cycle 6
CALL MAIN2
RETURN
Cycle 7
CALL FILTER
RETURN
Cycle 8
CALL FILTER
RETURN
Cycle 9
CALL FILTER
RETURN
Cycle 10
CALL DRUON1
RETURN

IN_OR_OUT

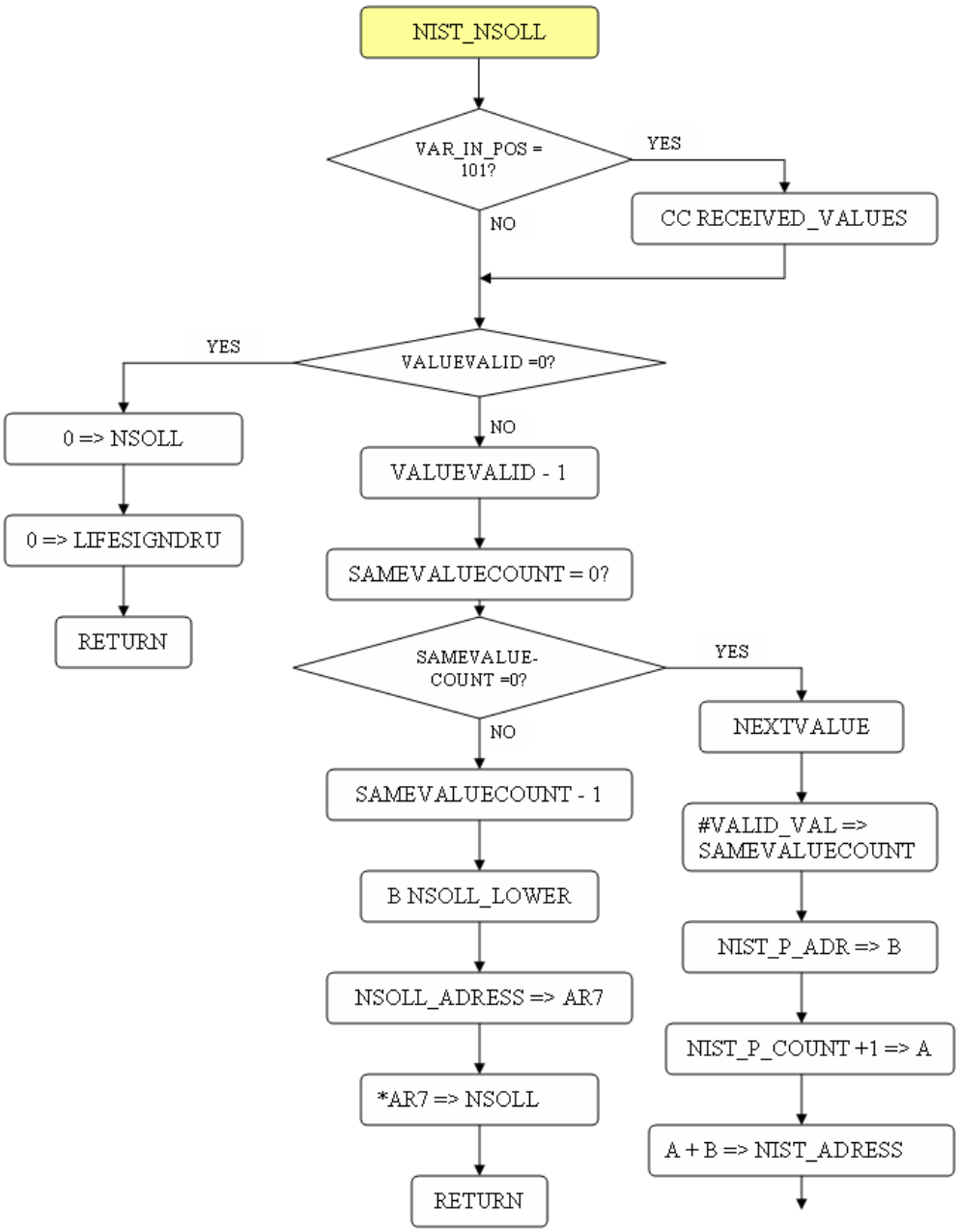

IN_OR_OUT
DATADIR = 1?
YES
NO
WRITE_DATA_CHECK
READ_DATA
SLEEPCOUNTER = F?
NO
YES
CALL COMMANDR
WRITE_DATA
0 => SLEEPCOUNTER
CALL OUTLIFEVAL
SLEEPCOUNTER + 1
RETURN

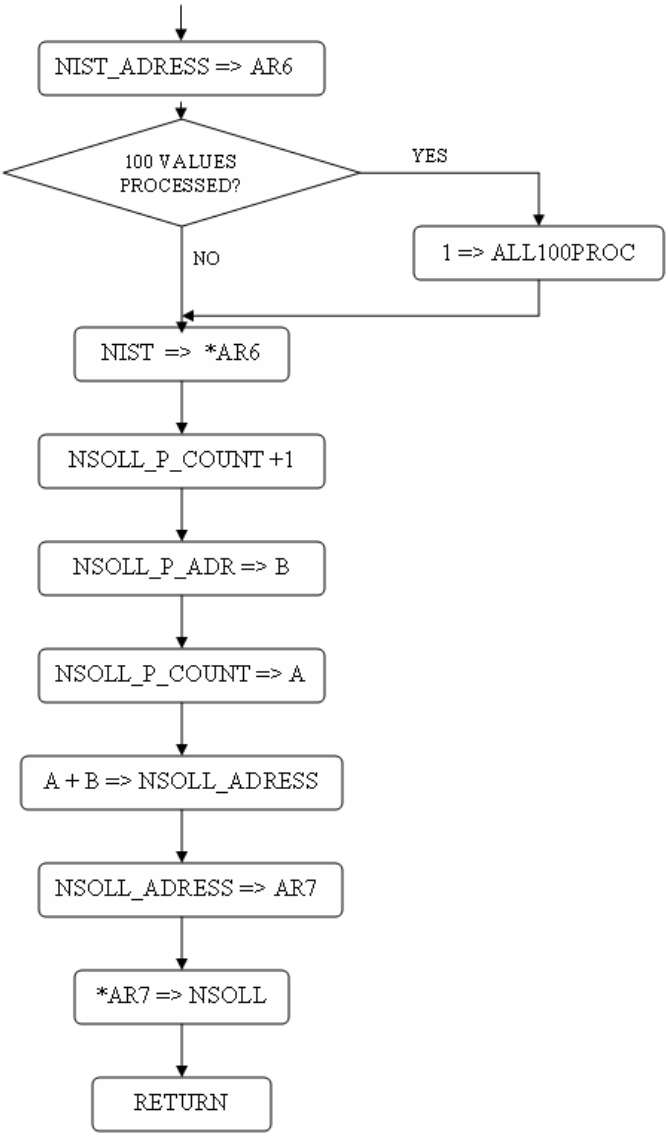
COMMANDR


COMMANDR
CALL VALUEREAD
A !=0?
NO
YES
HIGHBYTESTORE
CALL ASC2HEX
HEX => A
LOWVALUE => B
B*256 +A => WORDTEMP
#00h => HIGHBYTE
WORDTEMP => B
VALLOCK = 1?
YES
NO
ADR_COUNT_IN => A
VAR_IN_POS =>B
LOWBYTESTORE
CALL ASC2HEX
HEX => LOWVALUE
#01h => HIGHBYTE
RETURN
CHECK COMMAND

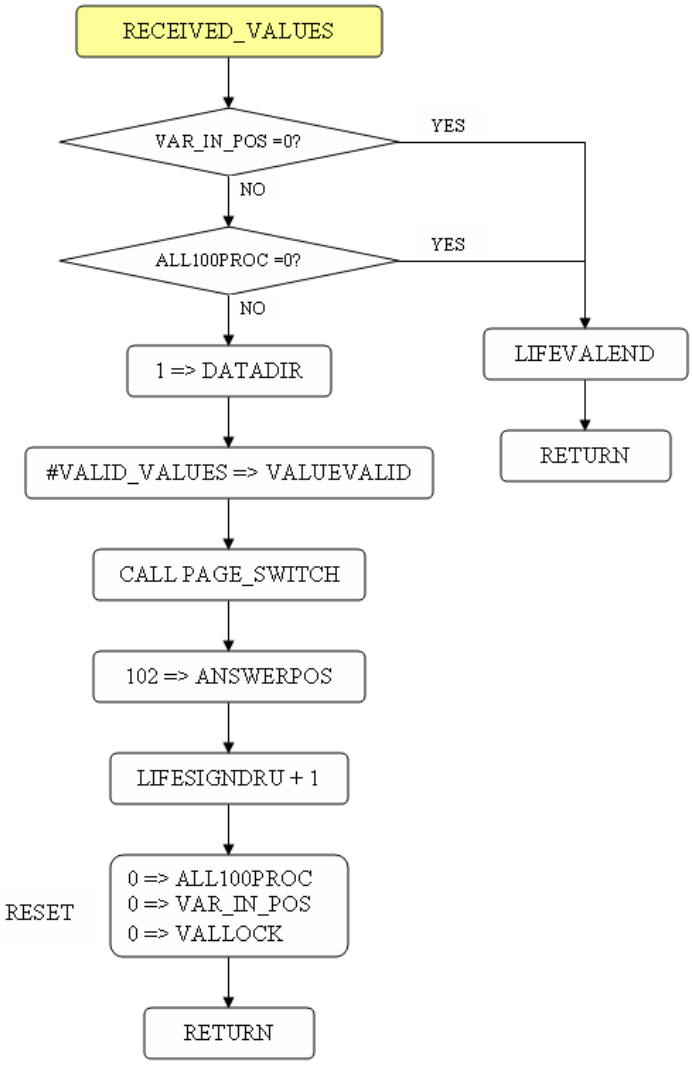

A + B => AR4
WORDTEMP => *AR4
VAR_IN_POS + 1
#LENGTH_CMDTAB - B
A ≥ 0?
NO
YES
UNKNOWN
COMMAND => A
CALL SENDHXWERT
#CMDTAB + B
ADRESS => HILF1
HILF1 => A
RETURN

NIST_NSOLL

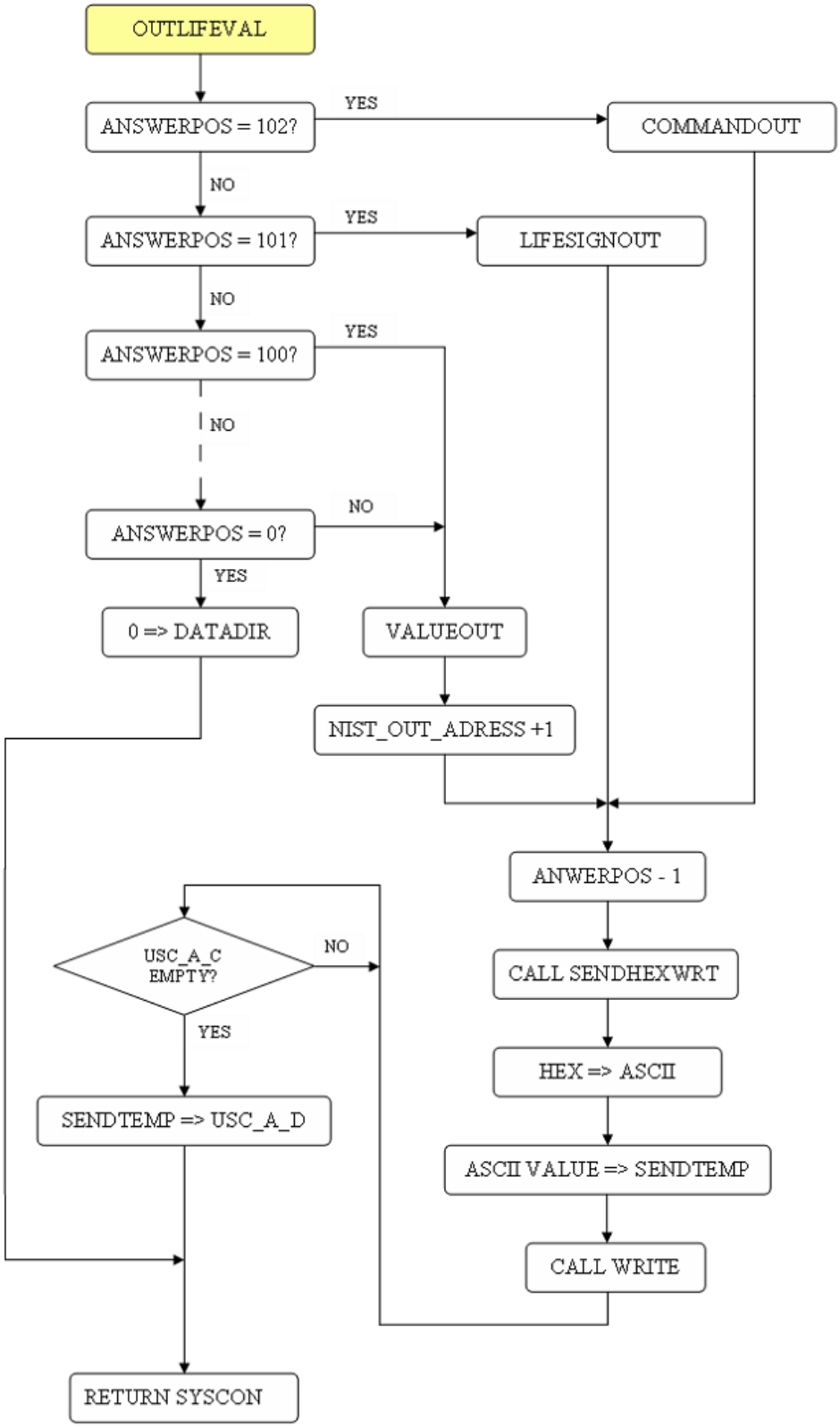

NIST_NSOLL
VAR_IN_POS = 101?
YES
NO
CC RECEIVED_VALUES
YES
VALUEVALID =0?
NO
0 => NSOLL
VALUEVALID - 1
0 => LIFESIGNDRU
SAMEVALUECOUNT = 0?
RETURN
SAMEVALUE-COUNT =0?
YES
NO
NEXTVALUE
SAMEVALUECOUNT - 1
#VALID_VAL => SAMEVALUECOUNT
B NSOLL_LOWER
NIST_P_ADR => B
NSOLL_ADRESS => AR7
NIST_P_COUNT +1 => A
*AR7 => NSOLL
A + B => NIST_ADRESS
RETURN


NIST_ADRESS => AR6
100 VALUES PROCESSED?
YES
NO
1 => ALL100PROC
NIST => *AR6
NSOLL_P_COUNT +1
NSOLL_P_ADR => B
NSOLL_P_COUNT => A
A + B => NSOLL_ADRESS
NSOLL_ADRESS => AR7
*AR7 => NSOLL
RETURN

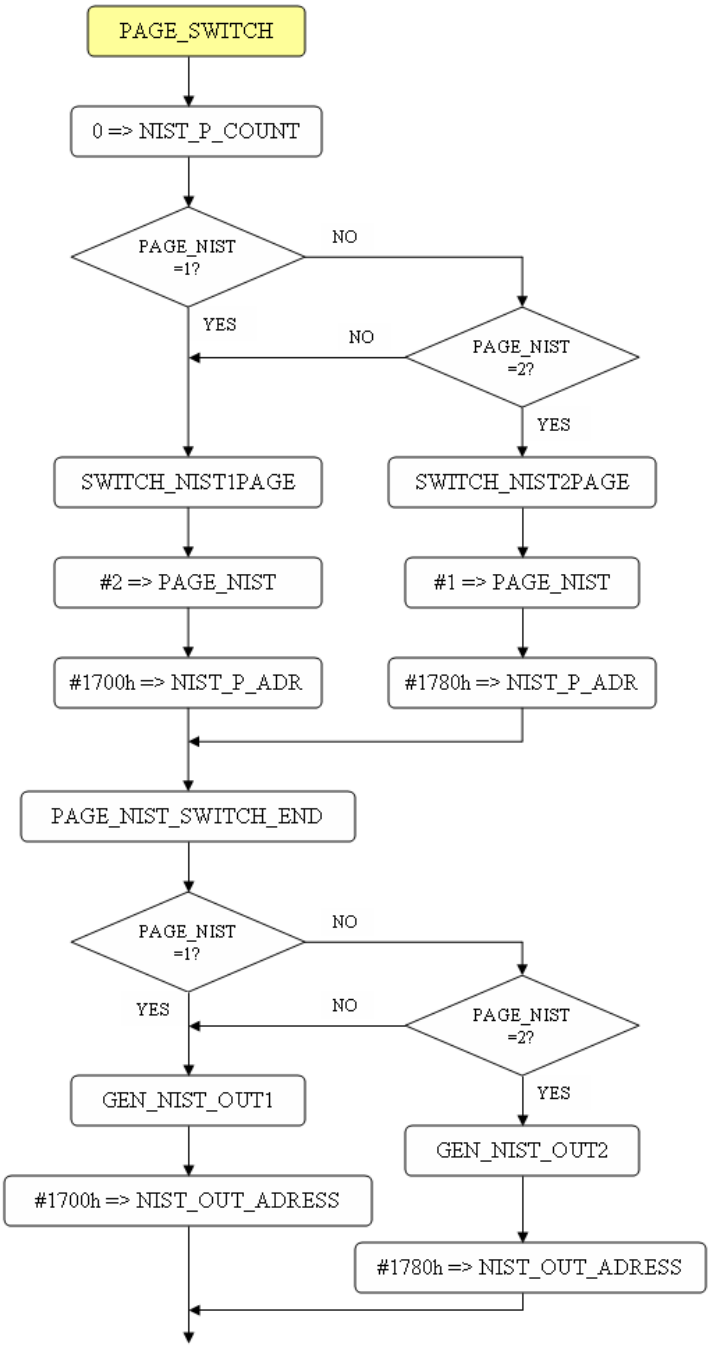
RECEIVED_VALUES

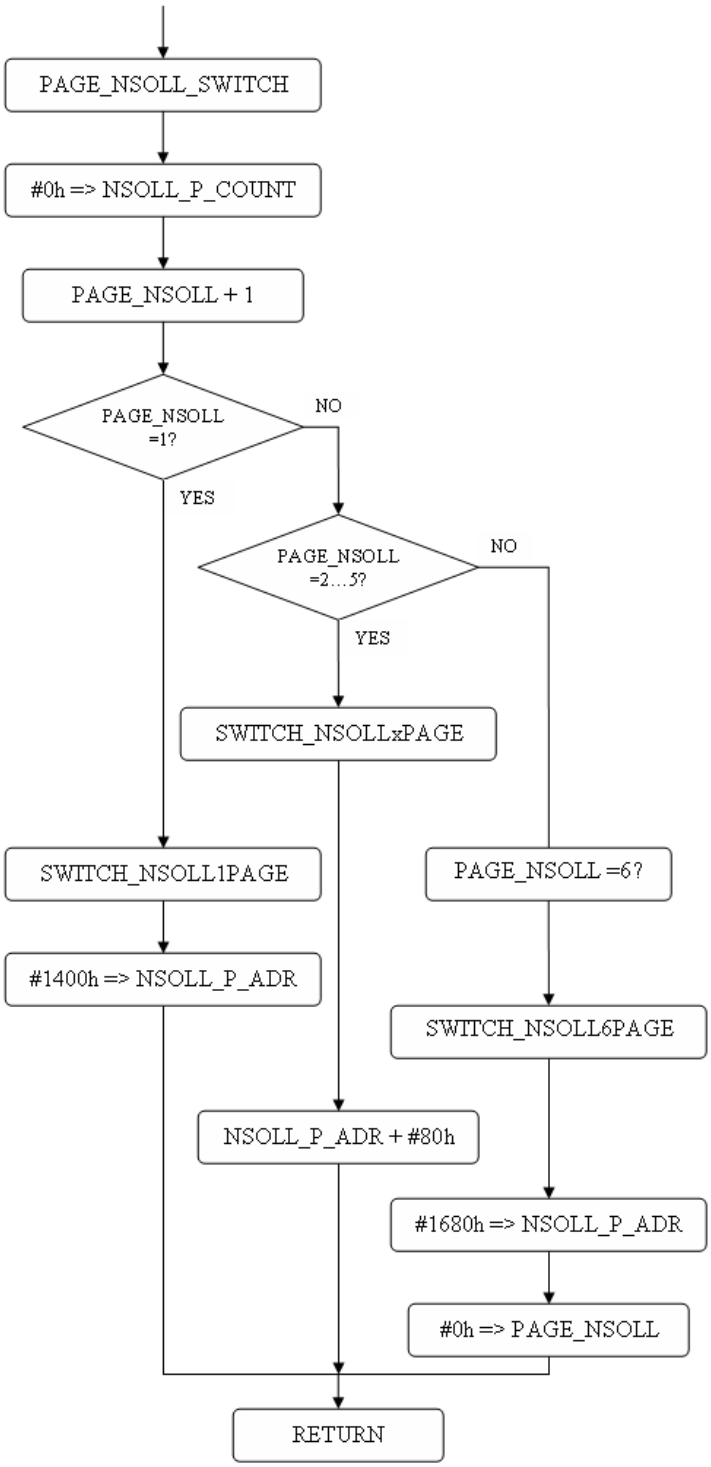


OUTLIFEVAL

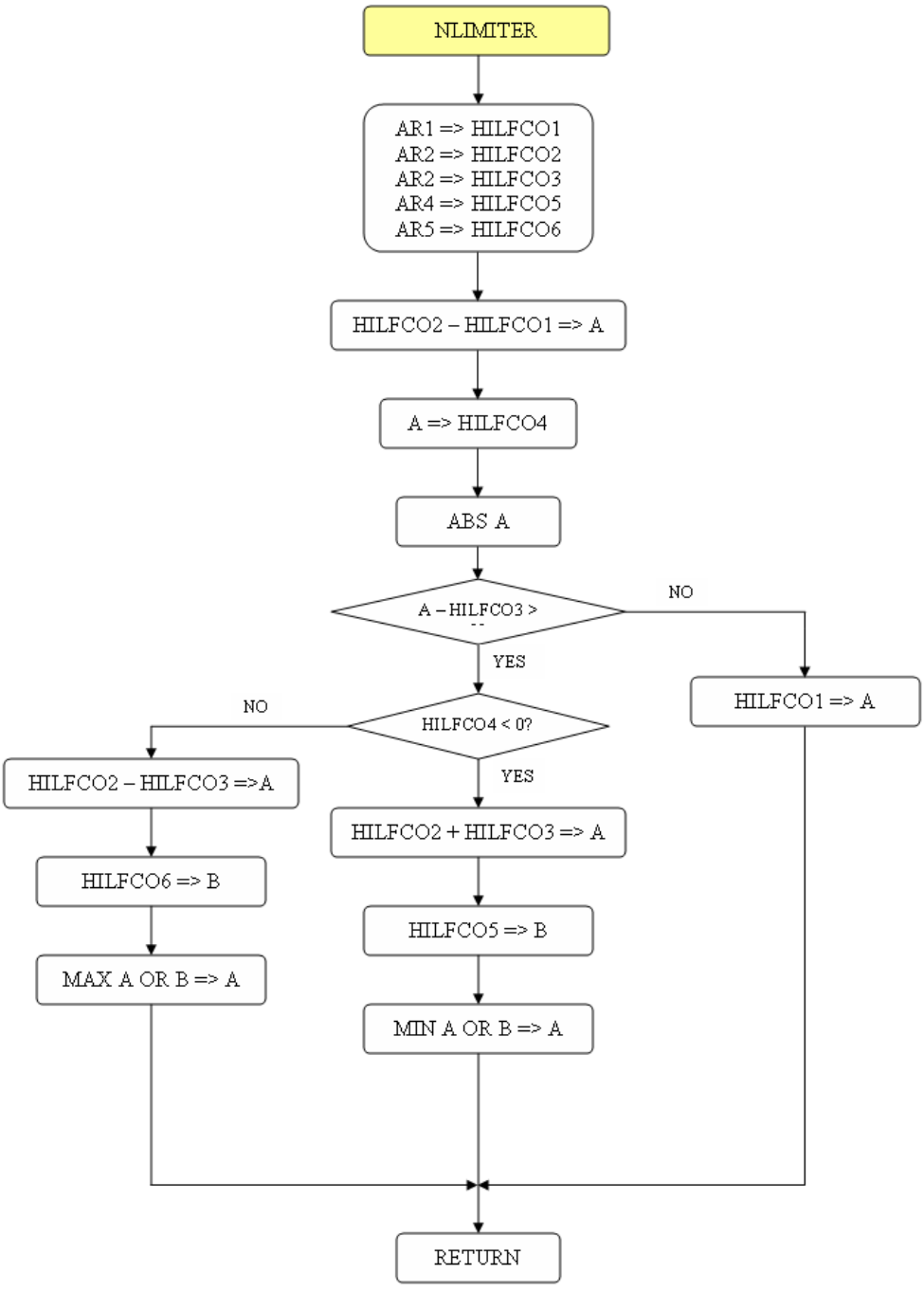

OUTLIFEVAL
ANSWERPOS = 102?
YES
COMMANDOUT
NO
ANSWERPOS = 101?
YES
LIFESIGNOUT
NO
ANSWERPOS = 100?
YES
NO
ANSWERPOS = 0?
NO
YES
0 => DATADIR
VALUEOUT
NIST_OUT_ADRESS +1
ANWERPOS - 1
USC_A_C EMPTY?
NO
YES
CALL SENDHEXWRT
HEX => ASCII
SENDTEMP => USC_A_D
ASCII VALUE => SENDTEMP
CALL WRITE
RETURN SYSCON

PAGE_SWITCH

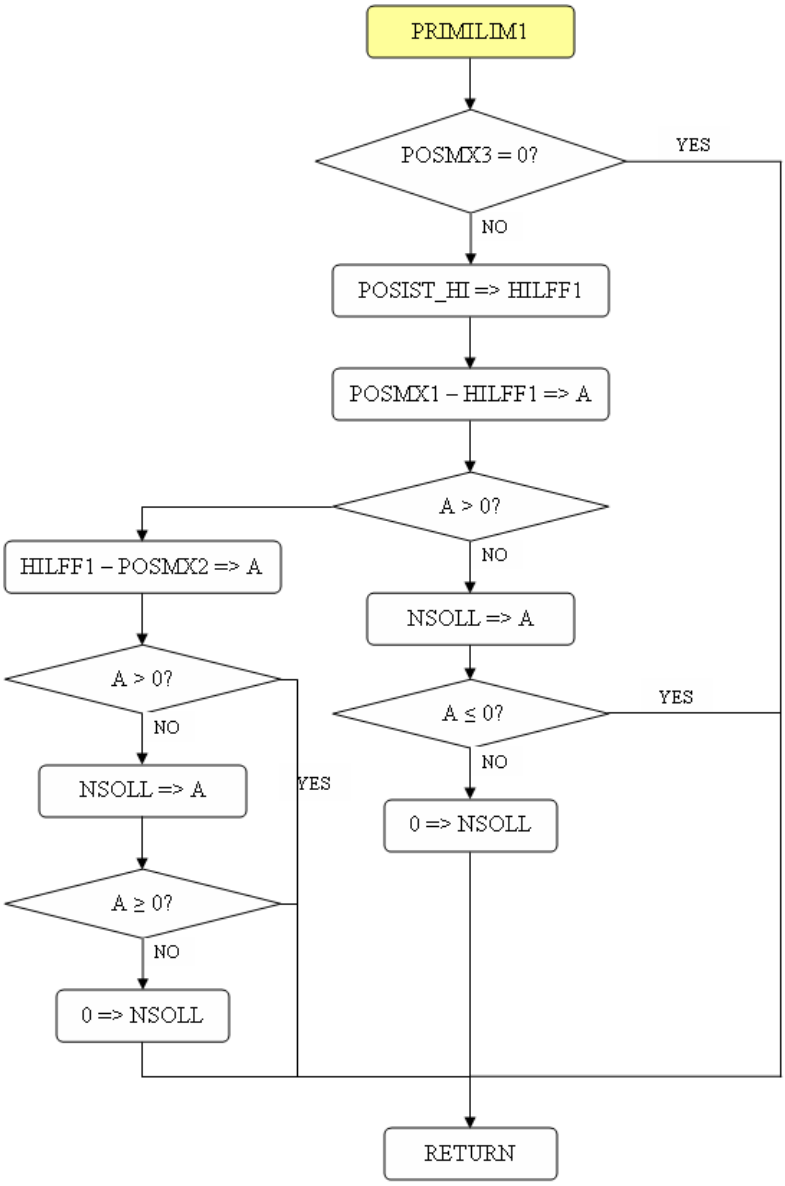

PAGE_SWITCH
0 => NIST_P_COUNT
PAGE_NIST =1?
NO
YES
NO
PAGE_NIST =2?
YES
SWITCH_NIST1PAGE
SWITCH_NIST2PAGE
#2 => PAGE_NIST
#1 => PAGE_NIST
#1700h => NIST_P_ADR
#1780h => NIST_P_ADR
PAGE_NIST_SWITCH_END
PAGE_NIST =1?
NO
YES
NO
PAGE_NIST =2?
YES
GEN_NIST_OUT1
GEN_NIST_OUT2
#1700h => NIST_OUT_ADRESS
#1780h => NIST_OUT_ADRESS


PAGE_NSOLL_SWITCH
#0h => NSOLL_P_COUNT
PAGE_NSOLL + 1
PAGE_NSOLL =1?
NO
YES
PAGE_NSOLL =2…5?
NO
YES
SWITCH_NSOLLxPAGE
SWITCH_NSOLL1PAGE
PAGE_NSOLL =6?
#1400h => NSOLL_P_ADR
SWITCH_NSOLL6PAGE
NSOLL_P_ADR + #80h
#1680h => NSOLL_P_ADR
#0h => PAGE_NSOLL
RETURN

NLIMITER


NLIMITER
AR1 => HILFCO1
AR2 => HILFCO2
AR2 => HILFCO3
AR4 => HILFCO5
AR5 => HILFCO6
HILFCO2 – HILFCO1 => A
A => HILFCO4
ABS A
A – HILFCO3 >
NO
YES
HILFCO1 => A
NO
HILFCO4 < 0?
YES
HILFCO2 – HILFCO3 =>A
HILFCO2 + HILFCO3 => A
HILFCO6 => B
HILFCO5 => B
MAX A OR B => A
MIN A OR B => A
RETURN

PRIMILIM


PRIMILIM1
POSMX3 = 0?
YES
NO
POSIST_HI => HILFF1
POSMX1 – HILFF1 => A
A > 0?
NO
HILFF1 – POSMX2 => A
NSOLL => A
A > 0?
A ≤ 0?
YES
NO
NO
NSOLL => A
YES
0 => NSOLL
A ≥ 0?
NO
0 => NSOLL
RETURN

Appendix B

Assembler code

As with the flow diagram only a few assembler code files will be presented in this appendix. The codes are a part of the company production secret and therefore only some basic files will be presented in appendix (B).

Nlimiter.asm **********

```
* MODUL NLIMITER.ASM
* ==================
*
* 19.12.05 adapted from LIMITER.ASM j.n
*
* This module regulates the MAX/MIN NSOLL Value
*
* INPUTS: AR1: NEW NSOLL INPUT VALUE
* AR2: LAST NSOLL OUTPUT VALUE
* AR3: MAX ALLOWED NSOLL INCREMENT
* AR4: UPPER NSOLL LIMIT
* AR5: LOWER NSOLL LIMIT
*
* OUTPUTS: ACCU: NEW NSOLL OUTPUT VALUE
* NOTE: in/out's are in lower 16 bits.
*
*************************************************************************
* INCLUDE CONSTANT DEFINITIONS
* ----------------------
.include "KONSTANT.ASM" ; Definition of all System constants
.sect "COMMONP"
NLIMITER
SSBX OVM
```

```
NOP
SSBX SXM
NOP
* save input parameters
LD #COMMONP, DP
LDM AR1, A
STL A,HILFCO1 ; Limiter input -> HILFCO1
LDM AR2, A
STL A,HILFCO2 ; previous limiter output -> HILFCO2
LDM AR3, A
STL A,HILFCO3 ; max step size -> HILFCO3 (positive value!)
LDM AR4, A
STL A,HILFCO5 ; upper Limit
LDM AR5, A
STL A,HILFCO6 ; lower Limit
* build difference
LD HILFCO2, 16, A
SUB HILFCO1, 16, A
STH A, HILFCO4 ; AltOut - NeuIn: HILFC02 - HILFCO1
* check if limitation necessary
ABS A
SUB HILFCO3, 16, A
BC HANDLIM09, ALT ; jump if change smaller then maxStep
* upper or lower limit?
LD HILFCO4, 16, A ; difference -> ACCU
BC HANDLIM01, ALT ; jump to upper limit handling
* lower limit handling
LD HILFCO2, 16, A
SUB HILFCO3, 16, A ; NeuOut:= AltOut-maxStep 02-03
LD A, -16, A ; shift to lower accu
LD HILFCO6, B
MAX A
B HANDLIMEND
* upper limit handling
HANDLIM01
LD HILFCO2, 16, A
ADD HILFCO3, 16, A ; NeuOut:= AltOut+maxStep
LD A, -16, A ; shift to lower accu
LD HILFCO5, B
MIN A
B HANDLIMEND
```

```
HANDLIM09
LD HILFCO1, A ; NeuIn = NeuOut -> ACCU No change of NSOLL
HANDLIMEND
RET
```

Primilim.asm

```
**********************************************************************
*
* MODUL PRIMILIM.ASM 07.06.99 R. Aregger
* ==================
* Version 1.0
*
* DAS MODUL PRIMILIM ENTHAELT DIE PRIMITIVE POSITION-
SGRENZUNG.
* VORAUSSETZUNGEN:
* INPUT: KEIN INPUT
* OUTPUT: KEIN OUTPUT
* UEBERGABE: INSOLL
* VERZWEIGUNGEN: KEINE
*
* ÄNDERUNGEN/NEUES:
* Adapted for stabilo project 2005.12.15 j.n
**********************************************************************
.sect "MAIN"
PRIMILIM1
**********************************************************************
* POSIST_HI -> HILFF1:
* ----------
LD #FUNCP, DP
LD POSMX3, A ; POSMX3 = 0000h -> Primilim disabled
RC AEQ
LD #STEGP, DP
LD POSIST_HI, A
LD #FUNCP, DP
STL A, HILFF1
* TEST, OB POSIST_HI > POSMAX:
* ----------------
LD POSMX1, A ; MAX
SUB HILFF1, A
LD #MAINP, DP
BC NXT1, AGT
LD NSOLL, A
RC ALEQ
ORM #VMAXbit, STATUS
ST #0, NSOLL
RET
* TEST, OB POSIST_HI < POSMIN:
```

```
* ----------------
NXT1 LD #FUNCP, DP ; MIN
LD HILFF1, A
SUB POSMX2, A
LD #MAINP, DP
RC AGT
LD NSOLL, A
RC AGEQ
ORM #VMAXbit, STATUS
ST #0, NSOLL
RET
*******************************************************************************
*******************************************************************************
.mmregs
```

Debug.asm

```
* Memory ring Buffer for debug
*
* 2006-02-06 j.n
*
* usage CALL DEBUG_BUFFER
* input ACCA
* output ACCA
**
*******************************************************************************
DEBUG_BUFFER_1
pshm BG ; * *
pshm BH ; * *
pshm BL ; * *
pshm AG ; * *
pshm AH ; * *
pshm AL ; * *
pshm ST1
pshm ST0 ; ST0 abspeichern
nop
nop
ld #FUNCP2, DP
rsbx 1,sxm ; SXM abschalten da Adr. > 16 Bit
nop
ld DEBUG_BUF_1,A
ld A,B
add #DEBUG_START_1, A ; Bufferstartadresse in ACCA 1800h
ld #BOOTPAGE, DP
stlm A, AR5
nop
nop
ld DEBUG_TEMP_1, A ; Debug value.
stl A, *AR5
ld #FUNCP2, DP
add #1,B
stl B, DEBUG_BUF_1 ; Buffer erhöhen "erhöhen = increment/increase"
sub #DEBUG_BUFLENGTH_1, B
bc WRITE_DEBUG_BUFEND_1, BLT
st #0, DEBUG_BUF_1 ; Ringbuffer schliessen
WRITE_DEBUG_BUFEND_1
ssbx 1,sxm
ld #0,A
```

```
popm ST0
popm ST1
popm AL ; * *
popm AH ; * *
popm AG ; * *
popm BL ; * *
popm BH ; * *
popm BG ; * *
RET
*******************************************************************************
* DEBUG BUFFER 2
*******************************************************************************
DEBUG_BUFFER_2
pshm BG ; * *
pshm BH ; * *
pshm BL ; * *
pshm ST1
pshm ST0 ; ST0 abspeichern
nop
nop
ld #FUNCP2, DP
rsbx 1,sxm ; SXM abschalten da Adr. > 16 Bit
nop
ld DEBUG_BUF_2,A
ld A,B
add #DEBUG_START_2, A ; Bufferstartadresse in ACCA 1C00h
ld #BOOTPAGE, DP
stlm A, AR5
nop
nop
ld DEBUG_TEMP_2, A ; Debug value.
stl A, *AR5
ld #FUNCP2, DP
add #1,B
stl B, DEBUG_BUF_2 ; Buffer erhöhen "erhöhen = increment/increase"
sub #DEBUG_BUFLENGTH_2, B
bc WRITE_DEBUG_BUFEND_2, BLT
st #0, DEBUG_BUF_2 ; Ringbuffer schliessen
WRITE_DEBUG_BUFEND_2
ssbx 1,sxm
ld #0,A
```

```
popm ST0
popm ST1
popm BL ; * *
popm BH ; * *
popm BG ; * *
RET
```